# Évolution de la suite logicielle FORMID pour la conception, l'exécution et le suivi de situations d'apprentissage à distance

Patrick Echterbille

**CONSERVATOIRE NATIONAL DES ARTS ET METIERS**

**CENTRE REGIONAL RHÔNE-ALPES**

**CENTRE D'ENSEIGNEMENT DE GRENOBLE**

---

MEMOIRE

présenté par **Patrick ECHTERBILLE**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en Informatique option Systèmes d'Information (ISI)

---

# Évolution de la suite logicielle FORMID pour la conception, l'exécution et le suivi de situations d'apprentissage à distance

Soutenu le 03 Juin 2014

---

JURY

Président : M. Eric GRESSIER-SOUDAN

Membres : M. Claude GENIER
M. Philippe GARRAUD
Mme. Viviane GUERAUD
Mme. Anne LEJEUNE

**CONSERVATOIRE NATIONAL DES ARTS ET METIERS**

**CENTRE REGIONAL RHÔNE-ALPES**

**CENTRE D'ENSEIGNEMENT DE GRENOBLE**

---

MEMOIRE

présenté par **Patrick ECHTERBILLE**

en vue d'obtenir

LE DIPLÔME D'INGENIEUR C.N.A.M.

en Informatique option Systèmes d'Information (ISI)

---

# Évolution de la suite logicielle FORMID pour la conception, l'exécution et le suivi de situations d'apprentissage à distance

Soutenu le 03 Juin 2014

---

Les travaux relatifs à ce mémoire ont été effectués dans le Laboratoire d'Informatique de Grenoble (LIG), Equipe MeTAH
sous la direction de Viviane GUERAUD et Anne LEJEUNE

Claude Genier
Enseignant CNAM

# Avant-propos

## Remerciements

J'exprime tout mon respect et mes remerciements à l'ensemble des membres du jury qui sont présents pour ma soutenance. Je sais que le temps consacré à ce type d'événement est important et il ne faut pas l'oublier.

Je remercie tout particulièrement mes tutrices, Mesdames Viviane Guéraud et Anne Lejeune, pour l'opportunité qu'elles m'ont offerte, pour leur disponibilité malgré leur agenda très chargé, pour leurs conseils tout au long au stage. J'ai été très sensible à leurs encouragements durant la période plus difficile de rédaction du mémoire. J'ai beaucoup apprécié la confiance qu'elles m'ont offerte sur les choix du projet.

Je salue toute l'équipe de MeTAH. Ils m'ont aimablement accueilli dans leur environnement et m'ont permis de découvrir le contexte et l'ambiance d'un laboratoire de recherche.

Un tout grand merci à mon collègue de bureau, Mohameth-François Sy, expert en ontologie, avec qui j'ai pu avoir des échanges constructifs sur la liaison entre FORMID et OntoPrax. L'année passée dans le même bureau nous a souvent permis de confronter le point de vue du monde de la recherche et de celui des entreprises privées.

Je remercie vivement Nadine Mandran pour sa valorisation de mon travail effectué sur les traces logicielles, pour sa relecture du chapitre s'y rapportant, je n'avais sans doute pas imaginé l'importance de ce travail pour les autres membres de l'équipe.

Je tiens à saluer mes responsables de CGI qui m'ont autorisé à prendre ce congé individuel de formation. Je remercie tous mes collègues de CGI pour les marques d'amitié qu'ils m'ont manifesté durant le stage.

Enfin, je dédie ce mémoire à ma femme, qui a toujours été là pour moi pendant ces études, qui a été ma première relectrice dans tous les projets ces cinq dernières années, qui m'a encouragé, supporté, conseillé, c'est grâce à elle si je suis arrivé au bout de ces études.

## Liste des acronymes

| | |
|---|---|
| AJAX | Asynchronous JavaScript And XML |
| API | Application Programming Interface |
| CNRS | Centre National de la Recherche Scientifique |
| CSS | Cascading Style Sheet |
| DOM | Document Object Model |
| DTD | Document Type Definition |
| EIAH | Environnements Informatiques pour l'Apprentissage Humain |
| FORMID | Formation Interactive à Distance |
| HTTP | HyperText Transfert Protocol |
| IDE | Integrated Development Environment |
| IHM | Interface Homme Machine |
| IMAG | Informatique et Mathématiques Appliquées de Grenoble |
| INP | Institut National Polytechnique |
| INRIA | Institut National de Recherche en Informatique et en Automatique |
| JAR | Java Archive |
| JDBC | Java DataBase Connectivity |
| JPA | Java Persistance API |
| JSON | JavaScript Object Notation |
| JVM | Java Virtual Machine |
| LDI | Learning Design Infrastructure |
| LIG | Laboratoire Informatique de Grenoble |
| MeTAH | Modèles et Technologies pour l'Apprentissage Humain |
| Mo | Mégaoctet |
| MVC | Modèle-Vue-Contrôleur |
| OPI | Objet Pédagogique Interactif |
| REST | REpresentational State Transfert |
| SFTP | Secure File Transfert Protocole |
| SPA | Single Page Application |
| SQL | Structured Query Language |
| SSL | Secure Sockets Layer |
| SVN | Subversion |
| SWF | ShockWave Flash |
| TCP | Transmission Control Protocol |
| TICE | Technologie de l'Information et de la Communication pour l'Enseignement |
| URL | Uniform Resource Locator |
| VM | Virtual Machine |
| WAR | Web Archive |
| XML | eXtensible Markup Language |
| XSL | eXtensible Stylesheet Language |

# Sommaire

# Liste des figures

## Liste des tableaux



## Liste des extraits de code

# Introduction

Ce mémoire porte sur l'évolution de la suite logicielle nommée FORMID, pour « **Form**ation **I**nteractive à **D**istance ». Cette application permet de concevoir, exécuter et suivre des situations d'apprentissage décrites par des scénarios pédagogiques.

Le logiciel a été développé en plusieurs temps. À la suite de différents travaux réalisés par des stagiaires ou doctorants, une grande partie des développements a été réalisée en 2007 par un chercheur de l'équipe et les améliorations qui ont suivi ont été faites à l'aide de stagiaires. De ce fait, FORMID a peu progressé au niveau technologique, l'accent des évolutions passées étant mis sur la correction de défauts et sur des adaptations pour correspondre à des besoins de recherche précis. Il en résulte de nombreux problèmes de compatibilité avec les nouvelles versions de Java et des risques de sécurité par rapport aux plateformes et navigateurs actuels. Une grande partie du stage a donc consisté à migrer l'existant vers une architecture plus actuelle, plus robuste, en supprimant au passage les défauts relevés.

Lors des expérimentations du logiciel dans des classes de collège et lycée, les chercheurs ont remarqué qu'une partie des productions des enseignants lors de la conception d'un scénario pédagogique, ne trouvait pas sa place dans la représentation formelle prévue par le logiciel. Un volet de ce stage a pris en compte ce constat et les éléments de réponse pour changer cela, afin d'étendre les possibilités d'édition des scénarios. Des aides à la conception ont été apportées grâce à la connexion à un référentiel des connaissances, et des éléments issus de ce référentiel sont insérés dans le scénario pour l'enrichir et le contextualiser.

Le stage s'est déroulé au sein du Laboratoire d'Informatique de Grenoble, et plus particulièrement dans l'équipe MeTAH. Nous allons maintenant détailler plus précisément ce contexte.

## A. Laboratoire d'Informatique de Grenoble

Le **L**aboratoire d'**I**nformatique de **G**renoble (LIG) est un laboratoire qui a pour projet scientifique « l'informatique ambiante et durable ». Il contribue au développement des aspects fondamentaux de la discipline (modèles, langages, méthodes, algorithmes) et développe une synergie entre les défis conceptuels, technologiques et sociétaux associés à cette thématique. Il a pour partenaires académiques le CNRS, Grenoble INP, l'INRIA, les universités Joseph Fourier, Stendhal et Pierre-Mendès-France. [LIG14]

Le laboratoire est composé de 22 équipes réparties en cinq axes thématiques :

- Génie des Logiciels et des Systèmes d'Information
- Méthodes Formelles, Modèles et Langages
- Systèmes Interactifs et Cognitifs
- Systèmes Répartis, Calcul Parallèle et Réseaux
- Traitement de Données et de Connaissances à Grande Échelle

## B. Équipe MeTAH

L'équipe MeTAH (**M**odèles **e**t **T**echnologies pour l'**A**pprentissage **H**umain) est composée de didacticiens et d'informaticiens, et elle travaille sur la question de la conception, du développement et des usages des Environnements Informatiques pour l'Apprentissage Humain (EIAH). L'équipe fait partie au sein du LIG de l'axe « Système Interactifs et Cognitifs ». [LIG14]

MeTAH se donne pour objectif de comprendre comment les dimensions éducatives (didactiques ou pédagogiques) et les usages peuvent être pris en compte dans :

- la conception d'artefacts informatiques techniques (micromondes, simulations, tuteurs intelligents, jeux pour l'apprentissage, environnements collaboratifs, …)
- la conception de descriptions calculables de leur utilisation (scénarios d'apprentissage, d'encadrement, etc.)
- la conception de modèles computationnels des connaissances épistémiques et didactiques et de fonctionnalités associées (mécanismes de rétroaction, supervision, etc.)

L'équipe comprend 19 membres permanents, essentiellement des enseignants-chercheurs (que nous désignerons sous le terme « chercheur » par la suite) et une vingtaine de doctorants, stagiaires ou post-docs.

## C. Suite logicielle FORMID

FORMID a pour objectif de travailler sur la conception, l'exécution et le suivi de situations d'apprentissage particulières. Celles-ci sont composées d'exercices en relation avec des simulations, qui permettent à l'apprenant de résoudre les problèmes posés, FORMID s'interfaçant avec la simulation pour suivre l'avancée de l'apprenant et le conseiller en fonction de ses actions [CAGNAT07]. Chaque exercice propose un but à atteindre décomposé en étapes de résolution.

FORMID est centré sur le concept de scénario pédagogique. Dans le cas de la suite FORMID, chaque scénario pédagogique décrit un exercice précis avec un but défini et non pas un module d'apprentissage global donné en plusieurs séances sur une période donnée. Le modèle de scénario proposé par FORMID pour décrire un exercice définit :

- Le déroulement de la situation prescrite à l'élève, réparti en étapes, qui contiennent un point de départ et un point d'arrivée. Une condition de validation permet de tester la complétion de l'étape avant de passer à la suivante. Des messages en cas de réussite ou d'échec peuvent y être joints.
- Ce qui doit être observé pendant le travail de l'élève pour lui donner des conseils adéquats. Ces situations remarquables donnent aussi des informations pertinentes au tuteur qui suit l'avancement de la classe.

FORMID permet donc aux enseignants de concevoir des scénarios pédagogiques en lien avec une simulation, aux apprenants de réaliser des exercices sur simulation avec des

retours d'information sur ses choix de résolution, et aux tuteurs de suivre l'avancement d'une classe. La particularité de FORMID par rapport aux logiciels présents sur le marché, est le suivi précis, en temps réel ou non, que le professeur peut faire de sa classe. Chaque contrôle de situation remarquable défini lors de la conception du scénario est affiché dans un écran de suivi du groupe, et le professeur (appelé tuteur lors du suivi d'un groupe) a la possibilité de se focaliser sur une étape de l'exercice et/ou sur un élève. Cela permet par exemple de placer un contrôle sur une erreur fréquente pour un énoncé, et de constater le nombre d'apprenants qui déclenchent ce contrôle.

## D. Problématique et enjeux du stage

Des expérimentations ont été réalisées sur FORMID dans des classes de collège et lycée, avec des enseignants de différents niveaux dans le but d'étudier l'adéquation des outils fournis aux enseignants, à la réalisation de leurs tâches d'auteur et de tuteur.

### Étude des tâches auteur :

Les chercheurs ont observé la façon dont un enseignant concevait son scénario pédagogique, que ce soit sur maquette papier ou ensuite dans le logiciel. Il a été constaté qu'une partie de ce que l'enseignant mettait par écrit ne trouvait pas sa place dans l'application, entrainant une perte indéniable de son expertise pédagogique et didactique. Des extensions du scénario ont été envisagées, avec comme proposition l'ajout d'éléments épistémiques, et la liaison avec une ontologie représentant les connaissances et les types de tâche qui permettent de les travailler, préconisés par une institution pour une discipline et un niveau d'apprentissage donnés.

Ces extensions devraient d'une part permettre d'accompagner l'auteur dans sa tâche de conception, grâce aux informations didactiques fournies par l'ontologie, et d'autre part faciliter la future réutilisation et adaptation des scénarios grâce à l'expertise embarquée.

### Étude des tâches tuteur :

Les chercheurs ont analysé la manière qu'ont les tuteurs de suivre une classe, pour en déduire des comportements systématiques. Des traces logicielles ont été mises en place pour détecter chaque action des tuteurs. En parallèle, il était demandé à chaque tuteur d'expliquer à voix haute la motivation de ses actions, son raisonnement et ses déductions. L'analyse quantitative des traces logicielles et l'analyse qualitative des verbalisations ont permis de fournir des résultats pour la recherche.

Les traces logicielles à fournir pour ces analyses sont très complexes. En effet, il est intéressant de comprendre les choix du tuteur. Pour cela, il faut, en plus d'enregistrer l'action, compléter la trace avec un maximum d'informations contextuelles. Par exemple, combien d'élèves étaient visibles à ce moment-là ? Où est-ce que l'élève sélectionné a déclenché l'activation de beaucoup de contrôles en erreur ? La façon de tracer ces contextes avait été décidée de façon un peu empirique. Les traces produites lors des expérimentations se sont révélées peu homogènes, non fiables sur certains aspects, souvent peu adaptées aux traitements à faire, et un gros travail de nettoyage a dû être

effectué avant de pouvoir les exploiter. Un travail de stage avait alors eu pour objet de revoir la production de traces tuteur vers un nouveau format et de proposer un programme de transformation de l'ancien format vers le nouveau. Le format proposé s'est avéré incomplet et malaisé, les traces produites ou transformées lacunaires, erronées.

Un objectif de ce stage a donc été de repenser de façon systématique le modèle des traces produites par le tuteur, de consolider le code pour obtenir des données fiables et homogènes afin de pouvoir baser des recherches dessus.

### Problèmes relevés lors des expérimentations :

Pendant les nombreuses expérimentations de FORMID, un ensemble de défauts a été relevé. Certains relèvent des technologies employées, des évolutions des plateformes, des restrictions de sécurité des postes de travail ou du réseau utilisé. D'autres ont une origine applicative résultant des différents stages effectués sur le logiciel, au cours desquels des ajouts de code répondant à des besoins précis ont impacté d'autres parties de l'application.

De plus, toutes les versions développées par chacun des stagiaires n'ont pas été fusionnées, ce qui amène des différences de comportement suivant la version utilisée. Il a donc été intéressant de proposer une solution d'industrialisation, pour apporter un maximum de corrections et centraliser les développements via un gestionnaire de sources. L'objectif est d'apporter un maximum de clarté pour assurer la pérennité du logiciel et la reprise aisée par les futurs intervenants.

## E. Plan du mémoire

Le premier chapitre de ce mémoire dresse un état de l'art de la suite logicielle FORMID. Les différents modules y sont détaillés, ainsi que les technologies utilisées. Les problèmes rencontrés lors des expérimentations passées sont listés, et les nouveaux besoins répertoriés.

Un second chapitre relate la mise en œuvre d'une nouvelle architecture pour FORMID. Les choix techniques y sont justifiés. Nous présentons les actions d'industrialisation qui ont été menées, la création d'un nouveau serveur applicatif, et la sélection de librairies JavaScript pour remanier les interfaces homme-machine.

Dans le troisième chapitre, nous abordons les développements liés à la migration de FORMID vers la nouvelle architecture. La composition du serveur applicatif est expliquée. Ensuite, module par module, nous décrivons les changements effectués, les points essentiels à relever et les écueils rencontrés.

Le dernier chapitre explique la spécification des nouveaux besoins. Chaque demande a été recueillie, analysée, et une solution a été proposée puis implémentée. Il s'agit de la formalisation des traces produites par les actions de suivi de groupe, de l'enrichissement des scénarios pédagogiques existants et de l'internationalisation de FORMID.

Enfin, nous terminons ce mémoire par une synthèse du travail réalisé, des perspectives et évolutions possibles pour le futur. Un bilan personnel faisant office de retour d'expérience clôture ce rapport.

# I. FORMID – État des lieux

FORMID a été créé par l'équipe ARCADE, une des ex-équipes constitutives de l'équipe MeTAH, il y a plus de dix ans. Nous allons d'abord présenter les différents modules, ensuite l'infrastructure et les technologies employées, les défauts relevés au cours des expérimentations, et finalement les évolutifs souhaités.

## A. Présentation du logiciel

La suite logicielle FORMID est composée de trois modules (Auteur, Élève, Tuteur), axés autour d'un élément central, le scénario pédagogique.

### 1. Scénario pédagogique

Le scénario est la représentation formelle d'un exercice, avec un objectif et des étapes à effectuer pour l'atteindre. Il contient toutes les informations nécessaires pour encadrer l'élève : consignes globales et par étape, conditions de validation, messages en cas de réussite ou d'erreur, situations à observer avec la définition des conditions permettant de les détecter et les messages associés. Les étapes sont ordonnées, avec un état initial et un état final, qui correspond à une situation de départ de l'étape suivante. Les validations et les contrôles (situations à observer) sont liés à une condition logique représentée en Java, avec les variables de la simulation associée.

```
return  U<50 && U*I<200;
```

*Extrait de code 1 : Exemple de condition de validation*

Le scénario est sauvegardé dans un format XML (cf. annexe D.), respectant une norme décrite dans un fichier DTD. Il est composé d'informations générales sur l'exercice, ensuite d'une liste d'étapes avec pour chacune ses consignes, son état final et initial, son critère de

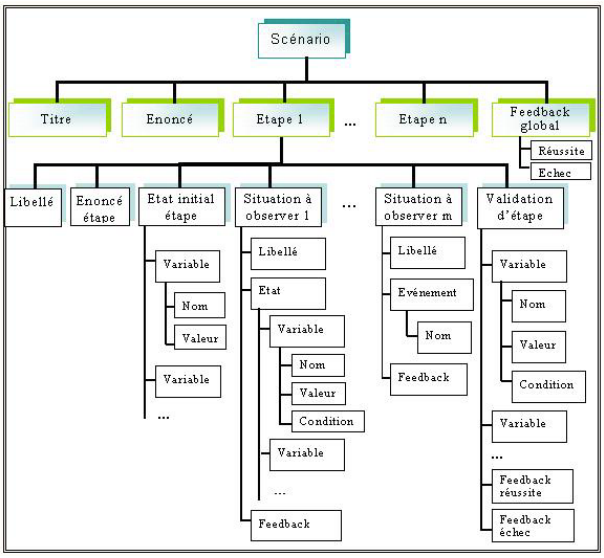



*Figure 1 : Représentation d'un scénario pédagogique sous forme d'arbre*

validation, et la liste des identifiants des éventuels contrôles à observer (cf. Figure 1).

Finalement on retrouve la liste des contrôles avec leur condition de déclenchement et les messages à afficher lors cet événement.

FORMID permet la compilation d'un scénario. Il s'agit d'une action de l'auteur permettant de générer le code Java afin de pouvoir l'exécuter dans le module élève. Le fichier XML est transformé par une feuille de style XSL qui en extrait les informations pertinentes pour créer un fichier de classe Java (cf. annexe E.). Celui-ci est passé au compilateur Java pour être converti en archive exécutable. S'il n'y a pas d'erreurs à la conception, cette action est transparente et instantanée. Par contre, si le compilateur lève des exceptions, l'utilisateur en est averti. De cette manière, le scénario joué à un groupe d'apprenants est garanti sans erreur quant à l'exécution du code Java (cela ne garantit en rien la cohérence des énoncés). Le module tuteur ne se sert que du fichier XML, il a besoin de l'enchainement des étapes et des contrôles du scénario décrit par l'auteur, mais pas d'exécuter le code compilé.

## 2. Module élève

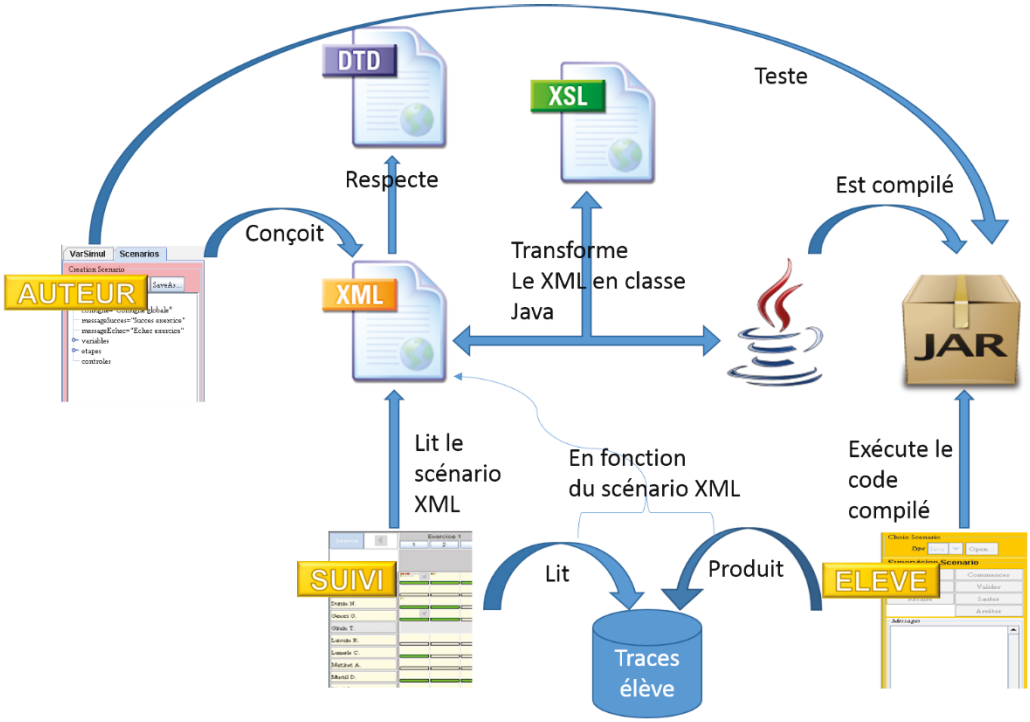



*Figure 2 : Vie d'un scénario dans FORMID*

Le module élève est hébergé sur une plateforme web, qui doit pouvoir permettre à l'apprenant de s'authentifier et de choisir un exercice. L'interface est chargée dans la même page que la simulation liée à l'exercice (cf. Figure 3).

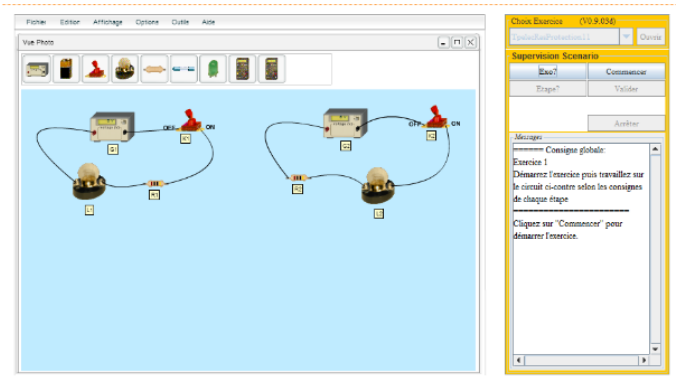


*Figure 3 : Écran de l'apprenant, avec la simulation à gauche et FORMID à droite*

La zone de message donne les consignes globales à l'exercice, et affiche par la suite les instructions par étape, et les alertes générées par des contrôles. L'apprenant peut dès lors interagir avec la simulation, en tenant compte des remarques qui apparaissent dans la zone de texte, et lorsqu'il pense avoir rempli les consignes de l'étape, il peut demander une validation. Le logiciel évalue alors l'état de la simulation, et approuve ou non la validation. Si l'étape est réussie, les consignes de l'étape suivante sont affichées ; si la simulation n'est pas dans un état valide, un message d'erreur avertit l'élève qu'il doit continuer à chercher. Des possibilités de sauter une étape ou de revenir à l'état initial ont été implémentées, elles sont actives en fonction des souhaits des auteurs de scénario et de la simulation utilisée.

Afin d'observer le travail de l'élève au sein d'une étape, le module FORMID interroge l'état de la simulation à intervalle régulier (400 ms) et teste si les valeurs des variables de la simulation déclenchent un des contrôles définis pour l'étape. On se rend compte qu'il est fondamental pour FORMID que les simulations utilisées soient capables d'être interrogées sur leur état interne.

Chaque information recueillie sur le travail de l'élève est transmise au serveur afin d'être enregistrée, et utilisée par le module tuteur pour le suivi. On trace donc les connexions / déconnexions, les demandes de validation et de consignes, les contrôles déclenchés, et l'état de la simulation pour toute demande de validation ou tout déclenchement de contrôle.

### 3. Module tuteur

Le module tuteur dans FORMID permet de suivre un groupe lors d'une séance. Une séance peut être composée de plusieurs exercices (et donc de plusieurs scénarios). Cela est possible via un fichier de propriétés posé sur le serveur (cf. chap I-B-3). Le tuteur peut suivre le groupe en temps réel ou de manière asynchrone. Cela permet de rejouer une séance pour en tirer des enseignements.

L'interface est composée de trois niveaux de précision, permettant d'avoir une vue globale du groupe et de son avancement, de l'état du groupe sur une étape d'un exercice donné, ou au niveau le plus fin, des événements successifs lors de la progression d'un élève sur une étape.

Lors de l'affichage global de la séance (cf. Figure 4), les élèves sont affichés sur la gauche et les exercices (avec leurs étapes) sont sur la zone supérieure du tableau. Les élèves sont sélectionnables via une case à cocher, ce qui permet de se focaliser sur un sous-groupe, les étapes sont représentées par des boutons, il n'est pas possible de sélectionner un sous-ensemble d'étapes.

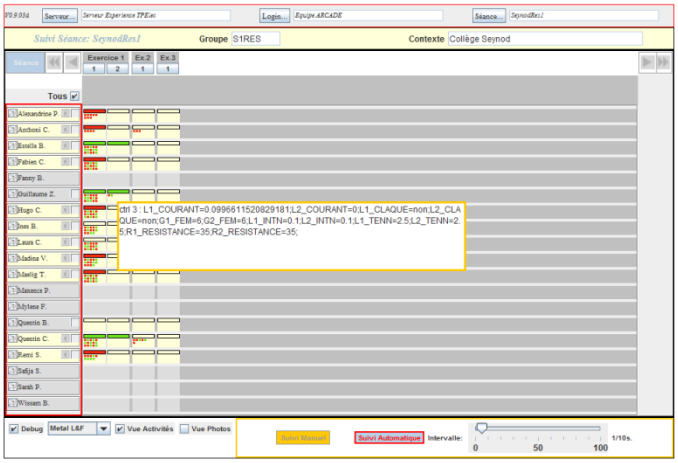



*Figure 4 : Écran de suivi global, avec survol d'un élément de validation*

Lorsqu'on sélectionne une étape, s'il y a au minimum deux élèves sélectionnés, on obtient le détail de l'étape (cf. Figure 5), c'est-à-dire une colonne par contrôle lié à celle-ci, et une colonne pour les demandes de validation. L'avancement ne s'affiche que pour les apprenants sélectionnés.

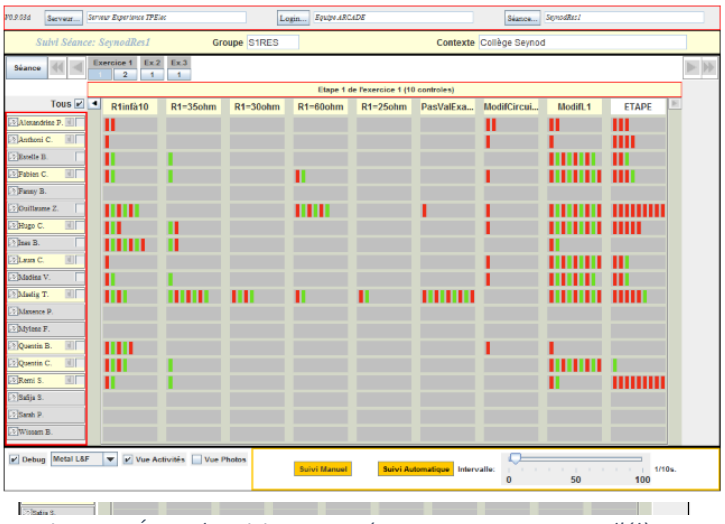



*Figure 5 : Écran de suivi pour une étape et un sous-groupe d'élèves*


*Figure 6 : Écran de suivi pour une étape et un élève*

Si un seul élève est sélectionné, la vue s'adapte (cf. Figure 6), chaque ligne représente maintenant une nouvelle trace de l'apprenant, ce qui permet de suivre la chronologie des évènements, entre les différents contrôles et les demandes de validation.

Le tuteur peut rapatrier les traces élèves soit manuellement, une trace à la fois, soit de manière automatique, suivant un intervalle choisi. Si le suivi s'effectue en temps réel, l'affichage se met à jour dès qu'une trace élève est produite. Si on rejoue la séance de manière asynchrone, on garde l'ordre d'apparition des événements mais on perd l'espacement entre deux événements. En effet, si l'intervalle est de 2 secondes, une donnée sera récupérée toutes les deux secondes, qu'il y ait eu une demi seconde ou dix minutes entre les deux événements élève.

Chaque donnée est typée en fonction de l'action de l'élève, et s'affiche de manière différente dans l'écran du tuteur. Chaque connexion/déconnexion joue sur la couleur de fond de la case élève, une demande de validation colorera le grand rectangle au croisement de la ligne de l'élève concerné et de l'étape en cours et chaque contrôle est représenté par un petit carré à ce même croisement. La couleur verte signifie une réussite, le rouge un échec, le jaune un élément neutre, tel qu'un élément de contexte de la simulation.

Le survol d'un élément de validation ou de contrôle ouvre une fenêtre de type « pop-up », qui affiche les caractéristiques de l'état dans lequel l'élève correspondant avait placé la simulation au moment de la production de la trace.

### 4. Module auteur

Le module auteur sert à concevoir un scénario, à le tester, à le compiler et le sauvegarder une fois terminé. Ce module peut être séparé des deux autres et tourner sur un autre serveur. En effet, son objectif est de produire un scénario XML et une archive Java associée. Ces fichiers pourront être déposés par la suite sur le serveur d'exécution de scénarios.

L'application permet à l'auteur de choisir entre deux modes, un mode création et un mode exécution, pour tester le résultat produit.

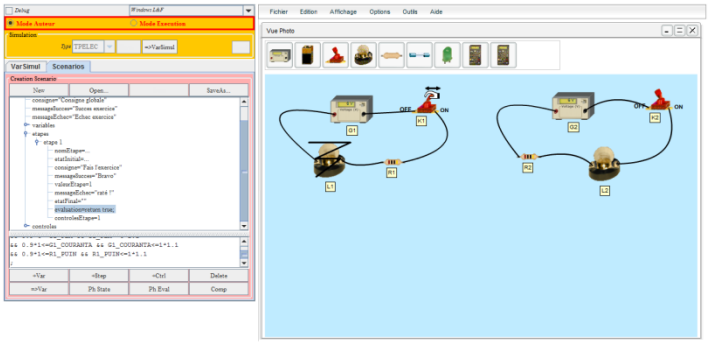


*Figure 7 : Écran de l'auteur, avec FORMID à gauche et la simulation à droite*

Le scénario est représenté par un arbre où chaque feuille est éditable. Certaines feuilles sont relatives à la simulation (variables de la simulation, états initiaux et finaux, condition de validation d'étape ou de déclenchement de contrôle). La communication entre le module Auteur et la simulation permet à l'auteur de se servir de la simulation pour enregistrer les variables pertinentes pour le scénario, ainsi que les états initiaux et finaux des étapes. L'auteur peut également sélectionner un sous-ensemble de variables, et obtenir sur demande une condition créée automatiquement avec les valeurs de celles-ci dans l'état courant de la simulation.

À partir de cette interface, il peut sauvegarder le scénario dans son format XML ou lancer une compilation pour obtenir le scénario sous forme d'archive exécutable.

## B. Technologie et infrastructures

FORMID est développé en Java, il est lancé sous forme d'Applet dans un navigateur internet. Il peut interagir avec des simulations, peu importe à priori la technologie de celles-ci. Dans les faits, FORMID a des interfaces vers le Java et le Flash. Le serveur « testbed » héberge l'application pour lui permettre d'être accessible sur internet.

### 1. Applet Java

Une Applet est un petit logiciel qui s'exécute dans un navigateur internet. En raison de la domination du langage Java dans ce domaine, on dit souvent « Applet Java » [Wikipedix13]. Le code compilé est chargé dans le navigateur web, et nécessite une machine virtuelle Java (JVM) installée sur le poste client pour exécuter le programme. L'intérêt est de décharger le serveur d'une grande partie du traitement, et de donner à l'utilisateur une impression de plus grande interactivité, car il n'y a pas de temps de latence dû au réseau.

L'interface graphique utilisée est Swing, ce qui donne aux applets un air désuet. De nombreuses failles de sécurité sont relevées chaque année, et les contraintes techniques nombreuses entrainent une grande utilisation de Flash (cf. chap. I-B-2) au dépend des Applets [COPE13].

### 2. Micromonde flash

FORMID est pensé pour pouvoir être utilisé avec n'importe quelle simulation. Techniquement, pour pouvoir être contrôlée par un scénario pédagogique, une simulation doit fournir un ensemble de services. Ces services permettent de contrôler la simulation lors de l'exécution du scénario : démarrer et arrêter la simulation, la mettre dans une certaine situation, recevoir de la simulation les informations permettant de tester les états de la simulation par rapport aux différentes situations définies dans le scénario [GUERAUD04].

Pour faire dialoguer FORMID avec des applications, des adaptateurs Java ont été développés. Un premier pour dialoguer avec d'autres Applets Java, un deuxième pour dialoguer avec un micromonde Flash portant sur les circuits électriques, TPElec [TPElec14].

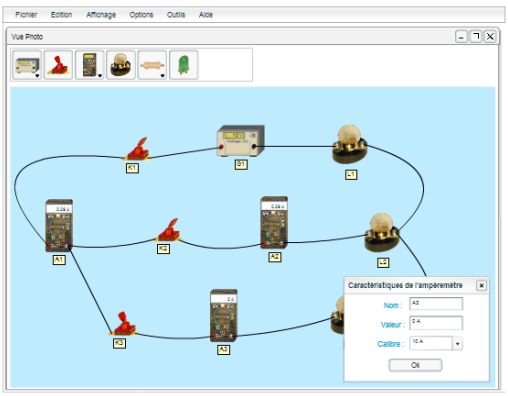



*Figure 8 : Micromonde TPElec*

La différence entre un micromonde et une simulation, c'est la possibilité pour l'utilisateur d'un micromonde de créer ou supprimer des objets de façon libre alors qu'une simulation permet de modifier uniquement les paramètres des objets existants.

TPElec permet de simuler des circuits électriques. Les lois physiques les régissant sont programmées et tiennent compte des caractéristiques des composants, caractéristiques modifiables par l'apprenant :

- choix de la résistance d'un récepteur,
- choix des calibres des voltmètres et ampèremètres,
- choix de la tension du générateur.

L'objet Flash est inséré sur la même page que FORMID, et l'adaptateur fait des appels JavaScript pour interagir avec le micromonde. La technologie Flash permet, via une librairie appelée ExternalCall [Adobe], de recevoir des appels et d'appeler des méthodes JavaScript. TPElec expose une méthode pour obtenir des renseignements sur son état, et appelle une méthode JavaScript qu'il faut enrichir pour correspondre au besoin. C'est-à-dire que la seule méthode d'appel JavaScript que TPElec comprend est « getInfo(*nomDeVariable)* ». Il prépare la réponse, et la renvoie en faisant l'appel à la méthode « TPElecToMoniteur(*reponse*) ». Il faut donc enrichir « TPElecToMoniteur » pour analyser la réponse (cf. Figure 9).

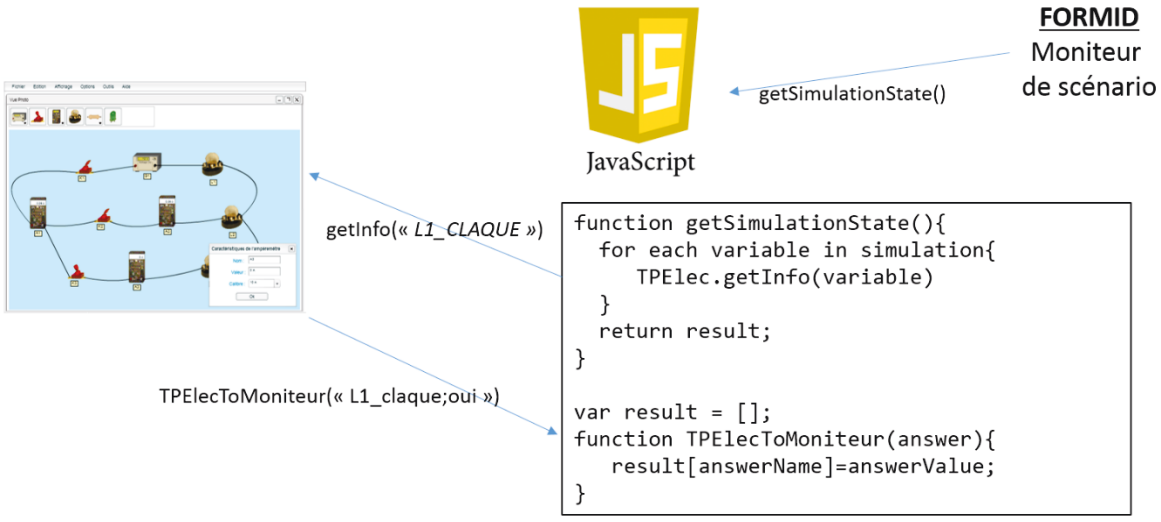



*Figure 9 : Communication entre FORMID et TPElec*

### 3. Plateforme testbed

Testbed est une plateforme web développée dans le cadre d'un projet de recherche européen (Plateforme Shared Virtual Lab de Kaleidoscope [noe-kaleidoscope14]). La plateforme contient LDI [MARTEL06] (« Learning Design Infrastructure ») permettant notamment la gestion d'utilisateurs, de séances de cours (enchainement d'exercices modélisés par un scénario), et la création de questionnaires à choix multiple pouvant être insérés dans les séances. Le serveur « testbeda0 » est situé dans les locaux du LIG, il n'y a actuellement que FORMID qui l'utilise.

## C. Anomalies identifiées

Les nombreuses expérimentations de FORMID ont permis aux chercheurs de relever un grand nombre d'anomalies, liées en partie à la technologie et à l'infrastructure, mais aussi aux développements et aux erreurs de code.

### 1. Compatibilité JVM

Pour exécuter une applet Java dans un navigateur, une machine virtuelle Java (JVM) doit être installée sur le poste client. C'est un petit programme gratuit, téléchargeable à la demande, qui exécute le code de l'applet. L'ordinateur hôte alloue une « sandbox » (bac à sable), zone cloisonnée de la mémoire pour empêcher d'accéder aux données locales.

Bien que la JVM soit installée sur une grande partie des ordinateurs dans le monde [Oracle], le comportement est rarement similaire d'un poste à l'autre. Le navigateur utilisé influence les résultats, et il n'est pas rare de devoir installer des versions différentes en fonction de celui-ci. Java publie régulièrement des mises à jour et des correctifs, en moyenne tous les deux mois [Oracle14], et suivant les configurations, certains postes vont se mettre à jour automatiquement ou non (cf. Figure 10). Les mises à jour entrainent souvent des comportements différents pour FORMID, certaines fonctionnalités de Java étant marquées comme dépassées, d'autres étant ré-implémentées.

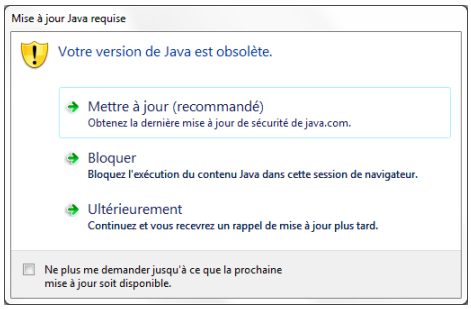



*Figure 10 : Mise à jour intrusive de Java*

Le fonctionnement intrinsèque des applets implique une exécution sur le poste client, il est donc impossible de garantir la version de la JVM utilisée par l'utilisateur.

### 2. Droit d'accès

Suite aux nombreuses failles de sécurité relevées ces dernières années dans les applets Java, Oracle a mis l'accent sur ces vulnérabilités lors des dernières versions de Java [KALLENBORN14]. Cela entraine des alertes de sécurité à chaque utilisation d'une applet Java non signée.

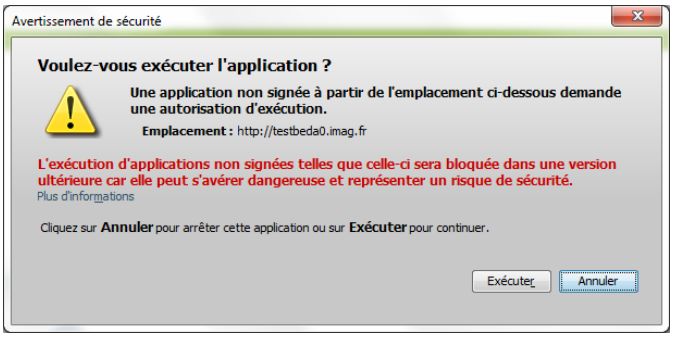



*Figure 11 : Alerte de sécurité Java*

Suivant les navigateurs, l'application peut être bloquée sans même un avertissement. Les paramètres de sécurité, réinitialisés à chaque mise à jour, ne peuvent être modifiés qu'avec des droits d'administration sur le poste. Cela pose de fait un problème récurrent lors de l'utilisation du module élève, les élèves n'ayant pas de droits d'administration l'exécution de FORMID peut être bloquée.

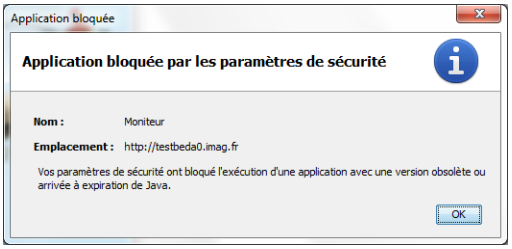



*Figure 12 : Logiciel bloqué par les paramètres de sécurité Java*

La signature d'une applet, pour limiter ces alertes de sécurité, est un processus complexe et le certificat généré n'est valide que 6 mois.

D'autres problèmes se posent au niveau du module auteur. Celui-ci n'est pas à disposition sur la plateforme testbed mais tourne de manière autonome sur un serveur local, « Mov'Amp » [COHEN07]. L'avantage de ce serveur web est de tenir sur une clé USB. Après l'avoir démarré, des pages HTML de lancement de l'outil sont disponibles pour accéder directement à un environnement avec une simulation pré-chargée. Chaque auteur peut donc reprendre le serveur chez lui pour travailler sur ses scénarios. Cependant, le module auteur a besoin de pouvoir charger des fichiers locaux vers l'application, et de sauvegarder le travail effectué vers un fichier XML sur le poste utilisateur. De plus, à la génération des classes Java, un certain nombre de fichiers compilés sont créés dans un répertoire. Comme les applets travaillent dans une « sandbox » (cf. chap. I-C-1), elles n'ont pas accès aux dossiers locaux. Pour modifier cela, il faut créer un fichier « JAVA.POLICY » à mettre à la racine du serveur. Ce fichier est complexe à mettre en œuvre et doit être modifié en fonction des répertoires dans lesquels on travaille.

```
//pour exécution poste Auteur dans Movamp
grant codeBase "http://localhost/pagesLancementLocal/-" {
 permission java.security.AllPermission;
};
permission java.io.FilePermission
"C:\\test\\_FormidTraces\\*", "read,write,delete";
```

*Extrait de code 2 : Exemple de réglages à mettre dans JAVA.POLICY*

### 3. Réseaux

Lors des expérimentations dans les collèges ou lycées, un problème récurrent est la gestion du réseau. En effet, en fonction du correspondant TICE de l'établissement, certaines règles sont fixées sur la gestion du pare-feu, et notamment sur les ports de communication ouverts vers l'extérieur. Traditionnellement, uniquement le port 80, par lequel transitent les appels web, est ouvert.

Cela ne pose pas de problème à l'ouverture de l'application, l'applet est chargée et peut s'exécuter de manière indépendante du serveur. Mais la base de données, dans laquelle

les traces de l'élève sont stockées pour pouvoir alimenter l'interface du tuteur, n'est accessible que sur le port 8080. La solution de contournement durant les expérimentations a été soit de faire ouvrir le port 8080 soit d'installer un serveur autonome sur l'ordinateur du tuteur. Comme celui-ci était sur le même réseau que les élèves, l'application pouvait fonctionner. Mais cette façon de procéder pose problème pour la manipulation et l'étude ultérieure des traces par les chercheurs.

#### 4. Anomalies logicielles

Une série d'anomalies ou de défauts d'ergonomie a été détectée au cours des différentes expérimentations. Certaines ont été corrigées dans une version mais n'ont pas été reportées partout, d'autres ont été contournées par des correctifs ad-hoc pour le besoin mais sans garantie de pérennité.

Parmi les principaux défauts identifiés, on trouve :

- Conversion entre affichage et stockage des données, en particulier pour les chaines de caractères en particulier. Pour l'utilisateur, il est plus lisible de voir les conditions de validation et les états de la simulation avec des retours à la ligne ou sous forme de tableau. Mais pour ne pas gêner la compilation, les caractères ajoutés pour l'affichage doivent être retirés.
- Problème lors du survol des zones de contrôle ou de validation dans l'écran de suivi. Une fenêtre pop-up doit afficher les données liées à l'aire survolée. Certaines fenêtres sont vides, ou avec des données incorrectes. Ce problème est apparu progressivement, suite à des mises à jour de Java.
- Problème de sélection des élèves. Pour sélectionner un élève, il faut au préalable désélectionner l'entièreté de la classe.

### D. Évolutifs demandés

Pour que les chercheurs puissent récolter et analyser les informations répondant à leurs questions de recherche, il est nécessaire que le logiciel puisse produire ces informations et que les données recueillies soient d'une qualité irréprochable.

#### 1. Amélioration des traces tuteur

Les traces tuteur enregistrent non seulement l'action d'un tuteur, mais aussi le contexte dans lequel s'effectue l'action, c'est-à-dire ce que voit le tuteur sur son écran et qui peut influencer son choix. Les premières expérimentations ont permis aux chercheurs de mieux cerner les informations pertinentes à tracer, celles qui avaient une valeur ajoutée dans le processus d'analyse des données enregistrées. Néanmoins, des problèmes d'harmonie et de fiabilité des traces ont été relevés et un travail de nettoyage a dû être effectué avant de pouvoir les exploiter.

Pour aller plus loin dans cette étude, les traces doivent être consolidées, pour garantir leur cohérence et ne plus avoir de travail de nettoyage à réaliser. Il faut également harmoniser les informations recueillies, le nommage des actions, les paramètres globaux à chacune des traces, de façon à pouvoir systématiser certains traitements et définir les spécificités par

action. Il faut déterminer quelles sont les informations pertinentes qui permettront d'effectuer des recherches sur les méthodes de suivi du tuteur. Finalement, il faut proposer un modèle de qualité pour garantir les performances, et des moyens de le vérifier.

Par la suite, il faudra étudier l'interfaçage de FORMID avec Undertracks [Undertracks14], une plateforme développée par l'équipe visant à recueillir les traces de différents logiciels, et à permettre de les exploiter via des briques d'analyse de données pour en extraire des tendances sous forme de tables et de graphiques.

### 2. Module auteur : lien avec une ontologie

Lors des expérimentations de conception de scénarios par des enseignants, les chercheurs ont constaté des différences entre le maquettage d'un scénario sur papier et sa modélisation électronique dans FORMID : une partie du savoir pédagogique et didactique du professeur ne trouvait pas sa place dans le modèle de scénario FORMID. Une étude menée en collaboration avec les didacticiens [CHAACHOUA10] de l'équipe montre que ce savoir peut être décrit à l'aide d'un cadre théorique fondé sur une approche praxéologique [CHEVALLARD06]. Les éléments praxéologiques sont divisés en types de tâche, technologies, techniques et erreurs fréquentes. On peut faire un parallèle entre ces éléments et ceux décrits par les enseignants lors du maquettage sur papier de leurs scénarios (cf. Tableau 1).

L'idée est alors d'étendre le modèle de scénario FORMID en y intégrant des éléments praxéologiques (type de tâche, technique, technologies, …) liés à l'organisation didactique de la connaissance à enseigner [GUIZANI12].

<table>
<tr><th>Document de conception fait par les enseignants</th><th colspan="2">Approche praxéologique</th></tr>
<tr><td>Problème posé<br>(ou sujet d'une étape d'un scénario)</td><td>T (type de tâche)</td><td rowspan="3">Praxéologie institutionnelle (du domaine de connaissances et du niveau d'apprentissage ciblés)</td></tr>
<tr><td>Résolution attendue</td><td>τ (technique permettant d'accomplir T)</td></tr>
<tr><td>Connaissances permettant de résoudre le problème</td><td>Θ (technologies justifiant τ)</td></tr>
<tr><td>Erreurs potentielles qu'il faudrait pouvoir détecter lors de l'exécution de l'activité par les apprenants</td><td colspan="2">Praxéologies personnelles a priori (erreurs fréquentes, …)</td></tr>
</table>

*Tableau 1 : Préparation d'une étape et approche praxéologique*

Une ontologie peut être définie comme étant une formalisation explicite [GUARINO95] et partagée [BORST97] des unités de sens (concepts) d'un domaine ainsi que leurs relations. Cette formalisation doit être indépendante d'un vocabulaire et d'occurrences donnés, être interprétable par un agent logiciel et respecter scrupuleusement la sémantique exprimée par un expert du domaine visé [GRUBER93].

Il existe dans l'équipe MeTAH un projet, OntoPrax, dont le sujet est la modélisation et la mise à disposition d'une ontologie concernant les savoirs dispensés dans l'enseignement. Cette ontologie qui représente des praxéologies se compose, entre autres, d'une partie noyau modélisant une praxéologie indépendamment d'une discipline donnée. Ce composant noyau est particulièrement important pour le travail présenté ici puisqu'il confère au référentiel un caractère générique permettant de l'étendre vers de nouvelles disciplines (voir IV-B).

L'objectif donc est de lier FORMID à ce projet pour guider la conception d'un scénario en se basant d'une part sur le programme institutionnel pour une discipline et un niveau donné, d'autre part sur les erreurs fréquentes à remarquer dans ce type d'exercice, et pour lesquelles l'enseignant pourrait créer un contrôle adéquat.

# II. FORMID – Refonte de l'architecture

Deux possibilités étaient envisageables pour FORMID, soit corriger autant que possible les problèmes, avec aucune garantie de la pérennité des solutions mises en œuvre, soit concevoir un nouveau modèle d'architecture, basé sur des technologies récentes et plus stables. Nous avons choisi cette deuxième solution, dont nous exposerons les avantages. Nous allons voir les détails de cette architecture technique, les actions d'industrialisation pour pérenniser le modèle, la création d'un serveur applicatif pour remplacer le logiciel tournant sur testbed et finalement la redéfinition de l'Interface Homme Machine (IHM).

## A. Proposition d'une nouvelle architecture

Aucune solution pérenne n'ayant été trouvée en restant sur une solution en Applet Java, nous avons revu :

- Le mode de communication entre l'interface et le serveur
- Le déplacement du moteur d'exécution du client vers le serveur
- La base de données et sa modélisation
- L'outil de traces système
- L'interface Homme Machine

### 1. Choix proposé

Au premier abord, avec le développement des nouvelles technologies et les interfaces web de plus en plus riches, FORMID donne une impression de logiciel vétuste. En effet, la librairie graphique Swing utilisée pour développer des IHM pour les applets, n'a jamais été mise au goût du jour et les composants datent de la fin des années 90. Des études de faisabilité ont été menées en début de stage, pour connaitre le coût d'une migration vers les technologies web actuelles, HTML5, CSS3 et des frameworks JavaScript pour rendre l'interface interactive.

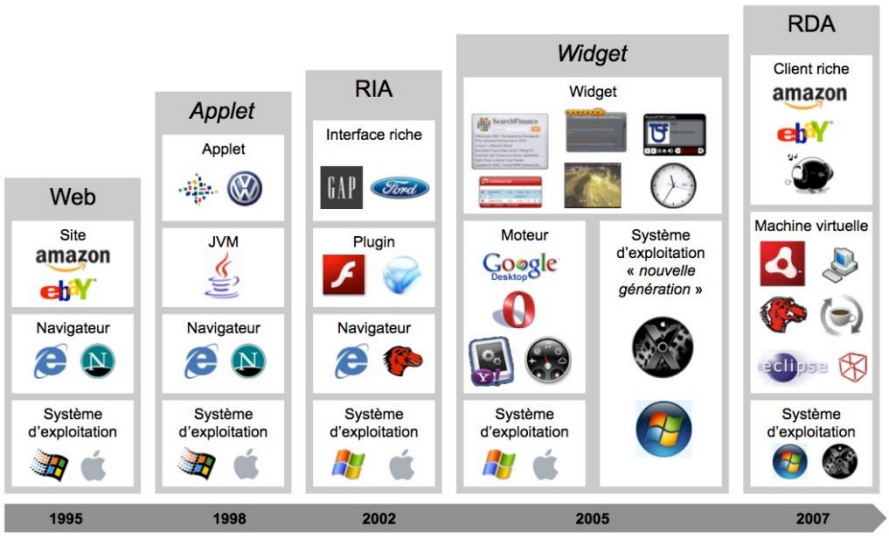



*Figure 13 : Évolution des interfaces web [CAVAZZA14]*

Afin de conserver le noyau de gestion de scénarios, il est apparu évident de garder le Java comme technologie principale. La bascule d'un mode entièrement client, intrinsèque aux applets Java, vers un mode client-serveur, format web standard, a amené à revoir les modes de communication entre l'interface et le moteur applicatif. En effet, le nombre d'appels au serveur est beaucoup plus conséquent, chaque action entrainant une requête transitant par le réseau. De plus, chaque appel doit transiter par le port 80, pour les raisons de sécurité vues au chapitre I-C-3.

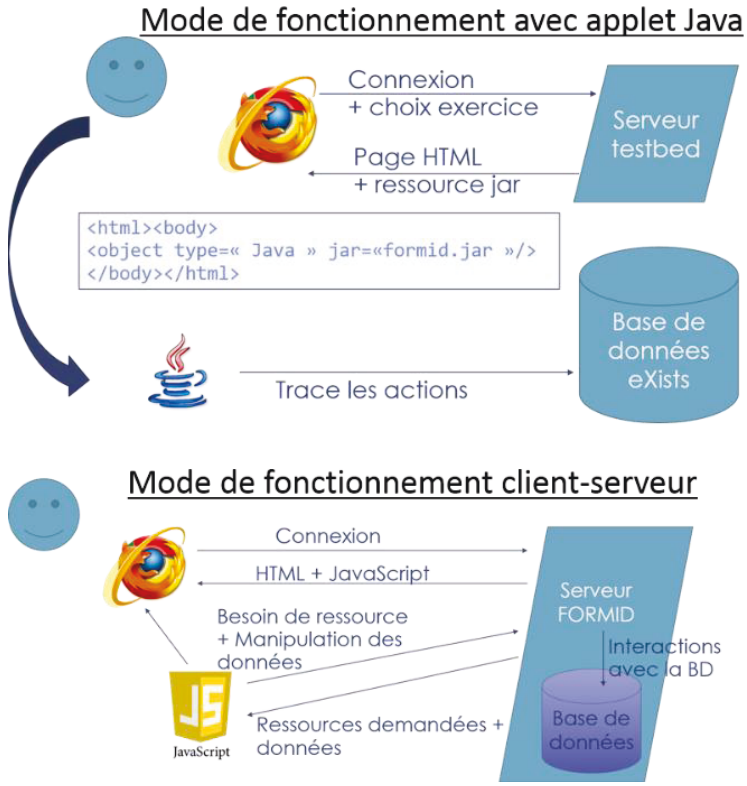



*Figure 14 : Mode de fonctionnement Applet Java vs. Client-serveur*

### 2. Avantages

L'architecture client-serveur a de nombreux avantages reconnus par la communauté, au niveau centralisation des données, sécurité, maintenance de logiciel [BEAUD14]. Par rapport à FORMID, on peut voir d'autres bénéfices spécifiques :

- Les mises à jour de Java ne sont plus un problème. En effet, le socle Java du logiciel est posé sur le serveur et n'est mis à jour que manuellement. L'application n'est plus dépendante de composants extérieurs, qui pourraient être mis à jour ou supprimés que ce soit par l'utilisateur ou de manière automatique et, de fait, empêcher le bon fonctionnement de FORMID.
- Les traitements qui nécessitent l'enregistrement de fichiers, tels que la génération de scénarios, se déroulent sur le serveur, il n'y a plus de problème d'accès au poste de l'utilisateur.

- On a une gestion plus fine du serveur, notamment quant à l'usage du réseau, car on peut fixer un point d'entrée central, sur le port 80, de façon à ce que FORMID soit utilisable par tous les établissements. La répartition par la suite entre les différents ports est transparente pour l'utilisateur.

#### 3. Inconvénients

Le principal inconvénient d'une architecture client-serveur est l'utilisation du réseau, celui-ci pouvant rapidement se retrouver saturé par les messages. Une attention particulière devra être portée aux performances, en évitant les messages superflus et en structurant les données pour n'échanger que le minimum nécessaire. De plus, si le réseau subit des perturbations, l'application peut être inutilisable. Néanmoins, si les problèmes sont situés sur un réseau extérieur à celui de l'établissement dans lequel se déroule l'expérimentation, il est toujours possible de lancer un serveur de backup sur un poste situé sur l'intranet du bâtiment.

Le serveur doit supporter une charge accrue, la majorité des traitements se faisant sur la plateforme centrale. Les tests n'ont montré aucune baisse de performances actuellement, mais il n'y a pas eu de mise en situation avec montée en charge importante. Si, lors des expérimentations futures, des ralentissements étaient constatés, l'achat d'un nouveau serveur adapté aux besoins devrait être envisagé.

Pour une expérience optimale de l'utilisateur, des librairies JavaScript sont utilisées. Celles-ci sont interprétées par le navigateur accédant à FORMID. Chaque type de navigateur (Internet Explorer, Firefox, Chrome, …) et chaque version de celui-ci ayant un moteur revisité d'exécution du JavaScript, les librairies compatibles actuellement avec la majorité des navigateurs ne le seront peut-être plus dans quelques années. Les librairies ont été testées avec les versions majoritairement utilisées pour le moment, mais certaines phases de test et mises à jour seront nécessaires si des comportements incohérents apparaissent dans les navigateurs du futur.

Ces inconvénients restent mineurs comparés aux nombreux avantages de la solution proposée.

### B. Industrialisation

Des faiblesses ont été constatées sur certains domaines transverses à tous les développements, hors du cadre spécifique de FORMID. Des actions d'industrialisation ont été menées afin d'offrir un environnement solide aux développements actuels et futurs.

#### 1. Mise en place d'un gestionnaire de sources

La suite logicielle FORMID existe depuis plus de dix ans, elle a été créée par des chercheurs avec une collaboration ponctuelle de stagiaires. Chaque intervenant a créé son dossier, avec ses sources plus ou moins documentées, sa gestion des versions, des présentations et des documents destinés à l'équipe ou à valider un stage. Ces dossiers ont été éparpillés dans le temps sur différentes machines ou disques durs.

Lors de l'arrivée d'un nouveau collaborateur, il était très difficile de lui donner l'entièreté des informations nécessaires, ou de lui dire quel dossier contient la version la plus récente

de tel module. Il est apparu évident qu'il fallait mettre en œuvre un gestionnaire de sources, au minimum pour le code, mais aussi pour les documents, afin d'avoir une centralisation des données.

### *a. Forge Imag*

Le laboratoire a accès à une forge [Forge IMAG14], c'est-à-dire un système de gestion de développement collaboratif de logiciel. Elle est basée sur le logiciel FusionForge [FusionForge14]. Elle permet la gestion des sources avec des outils au choix (SVN, Git, …), mais aussi le suivi des demandes d'évolutions et des rapports d'anomalies, la création de forums et wiki liés, une gestion de listes de diffusion, …

Le projet a donc été introduit sur cette forge, avec des autorisations d'accès très restreintes, pour garder la main sur le code source. Chaque nouvel intervenant devra créer son compte, puis être rajouté aux personnes ayant accès au projet par les chercheurs en charge de FORMID.

La gestion documentaire de la forge ne permet pas de créer des versions d'un même document, elle pourra être utilisée pour y mettre des modes d'emploi ou des articles publiés, mais il vaut mieux utiliser les gestionnaires de sources pour y placer les documents en cours de rédaction ou amenés à évoluer régulièrement.

### *b. Protocole GIT*

Pour la gestion des sources, une des inquiétudes principales des chercheurs était l'éventualité d'une fermeture de la forge et la perte des données associées. Pour cette raison, le protocole Git [Git --everything-is-local14] a été sélectionné. Contrairement aux logiciels de version classiques tels que SVN (Subversion), Git ne repose pas sur un serveur centralisé. Cela signifie que tout l'historique du projet est stocké sur chacun des postes développeur. Si la forge est inaccessible, l'entièreté du projet avec l'historique des changements est accessible en local (depuis la dernière demande de tirage). Dans la théorie, on peut échanger les informations directement entre postes développeur (cf. Figure 15). En pratique on garde toujours un serveur de référence. [Veille technologique14]

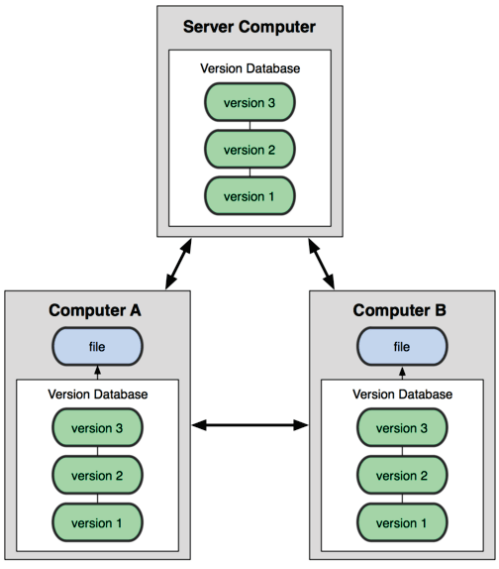



*Figure 15 : Gestionnaire de versions décentralisé*

Il est dès lors facile de créer une branche, de travailler dessus en local, de créer ses propres versions, puis de pousser les changements avec leur historique. Toutes les fonctions traditionnelles d'un gestionnaire de sources sont disponibles, la seule particularité pour le développeur est de devoir pousser ses modifications deux fois, une fois sur son gestionnaire local, une fois vers le serveur de la forge.

La communication entre le serveur de la forge et le poste local se fait via SSL (Secure Sockets Layer), protocole de sécurisation des données. Ce mécanisme fonctionne par authentification via clé publique / clé privée, il faut donc générer un jeu de clés, sauvegarder sa clé privée et faire connaitre sa clé publique au serveur.

La génération des clés se fait avec l'outil « Puttygen » de la suite logicielle gratuite Putty [Putty14]. Dans le paramétrage de son compte personnel sur la forge, un emplacement est prévu pour entrer les clés publiques, il peut y en avoir plusieurs (dans le cas de postes de travail différents par exemple). Un script tourne sur le serveur toutes les heures pour mettre à jour les clés des utilisateurs.


*Figure 16 : Modification des clés publiques dans la forge IMAG*

## 2. Amélioration des traces système

Les traces système sont les messages générés par le programme pour enregistrer les erreurs survenues, pour suivre les étapes d'un traitement, ou pour signaler les valeurs de variables ou de paramètres à un moment précis du processus. Elles n'ont un sens véritable que pour le développeur du logiciel. Il ne faut pas les confondre avec les traces fonctionnelles qui correspondent à un enregistrement d'événement utilisateur, ayant un sens pour le chercheur, et qui serviront pour l'analyse de l'utilisation du logiciel.

Les Applets Java s'exécutant sur le poste client, les traces étaient éphémères, puisqu'elles apparaissaient dans la console Java de la JVM de l'utilisateur. Lorsqu'un problème était rencontré, ce dernier faisait un copier-coller du texte présent dans la console et l'envoyait aux chercheurs. Si pour une raison quelconque (inattention, défaillance de l'ordinateur, …), l'utilisateur ne copiait pas la trace, elle était perdue. De plus, lorsque tout se passait bien, aucune trace n'était remontée par les utilisateurs.

La console de la JVM étant une zone de texte, les traces ne contenaient que les messages bruts, écrits par les développeurs. Aucune notion de catégorisation, de temps, ou d'identification n'était présente.

Le déplacement du traitement vers le serveur a permis de mettre en place une librairie standard fournie par le serveur applicatif, « JBoss Logging ». Elle fournit les méthodes classiques pour enregistrer des traces, et pour effectuer des redirections de sortie.

Les traces sont catégorisées par niveau, principalement :

- ERROR : Pour toute exception levée.
- INFO : Pour suivre des informations importantes dans le traitement
- DEBUG : Toute trace qui peut aider au développement, à valider un processus, …

Les sorties configurées dans FORMID sont au nombre de trois :

- La console : pour suivre en temps réel tout ce qui se passe sur le serveur. Les traces sont éphémères. Tout est affiché, y compris les traces produites par les librairies utilisées par le serveur.
- Le fichier server.log : Un sous-ensemble de traces s'y retrouve. Toutes les traces générées par FORMID, mais aussi celles générées par le serveur web et par la base de données. Les autres modules utilisés par le serveur applicatif sont exclus de ce fichier. Ce fichier est prévu pour être remplacé toutes les heures, c'est-à-dire qu'un nouveau fichier est créé et que l'ancien est renommé avec une extension « AAAA-MM-JJ-HH ».
- Le fichier formid.log : Seules les traces ayant été générées par FORMID s'y retrouvent. Le fichier est remplacé chaque demi-journée (extension : « AAAA-MM-JJ-AM/PM »).

Chaque trace est sous la forme :

« 09:11:00,009 DEBUG [fr.imag.lig.metah.formidweb.WebServer.LoginServer] Demande de connexion : {"userId":1,"token":"ECHTERPAT:139755595:737a39346a3"} », on retrouve la date (sous forme de « temps Unix », nombre de millisecondes depuis 1970 [DERKSEN14]), le mode de traçage, la classe ayant généré la trace, et le message.

Par rapport aux informations recueillies dans la JVM, il a été remarqué qu'on perdait un renseignement précieux : l'utilisateur ayant produit la trace. Si un professeur appelle pour signaler un problème, sa trace sera noyée dans les autres et l'investigation sera difficile à mener. La librairie a donc été étendue par une classe « FormidLog », qui enrichit chaque méthode d'un nouveau paramètre de type *String* : « userName », le nom de l'utilisateur. S'il est renseigné lors de l'appel à la méthode, il apparaitra avant le message dans la trace.

Cela a impliqué de récupérer à chaque point d'entrée du logiciel le nom de l'utilisateur connecté, et de le passer en paramètre de chacune des méthodes utilisées, afin que chaque trace du logiciel puisse utiliser cette information.

Enfin, sur une application web, toutes les erreurs JavaScript ne sont pas tracées sur le serveur, le développeur n'est donc pas au courant des problèmes survenus et de leur

occurrence. Une méthode générique JavaScript a donc été implémentée, pour envoyer au serveur des traces, qui seront enregistrées avec les autres et seront différenciables par un préfixe « JAVASCRIPT *** ». Cet appel au serveur sera non bloquant, l'interface n'attendra pas de réponse du serveur et pourra continuer à travailler une fois le message envoyé.

### 3. Installation d'une machine virtuelle

La nouvelle architecture n'est pas compatible avec le logiciel LDI installé sur le serveur testbed. Il ne permet que la mise à disposition de ressources web, telles que l'archive jar (Java Archive) contenant FORMID ou le fichier swf (ShockWave Flash) faisant tourner TPElec. La mise en œuvre de la nouvelle architecture implique la possibilité de déployer des paquets war (Web Archive) sur un serveur applicatif capable d'exécuter le code Java et traiter les appels web pour mettre à disposition les ressources.

Le serveur testbed a été déclaré vétuste par les administrateurs système et il n'y a plus que FORMID qui tourne dessus. Il a donc été décidé de laisser l'ancienne version de FORMID tourner dessus en attendant de valider complétement la nouvelle version, et de déployer un nouveau serveur pour la version client-serveur de FORMID.

Pour une question évidente de réduction des coûts, le LIG préconise l'utilisation de machines virtuelles (VM) sur un serveur commun. Les VM peuvent être taillées selon les besoins du projet. Pour FORMID, nous avons choisi un système d'exploitation basé sur Unix, CentOS [CentOS14], réputé pour sa stabilité. Il n'y aura pas de mises à jour automatique, pour être sûr de ne pas impacter le projet. Les accès se feront via Putty [Putty14] pour l'administration, et par SFTP pour le dépôt de fichiers, en utilisant des outils tels que FileZilla [FileZilla].

Le serveur applicatif et la base de données ont été installés. Un alias « formid.imag.fr » a été créé et est accessible de l'extérieur. Pour une question de facilité d'utilisation, des services ont été créés pour démarrer ou arrêter les composants. Un serveur web Apache a été configuré pour réceptionner tous les appels sur le port 80 et les rediriger vers le port 8080. Un module complémentaire est installé pour faire cette opération pour les websockets (cf. chap. III-A-1-d).

Pour faciliter le dépôt de la base de données, un dossier virtuel a été placé à la racine du serveur, il permet d'accéder directement au sous-dossier contenant le fichier stockant les données de la base. Des scripts de démarrage rapide du serveur applicatif et de la base de données ont été rédigés.

Par choix des administrateurs système, le droit d'effectuer des actions considérées dangereuses n'est possible qu'avec le mode « substitute user do », sudo. Cela concerne :

- les actions de création et suppression de fichiers/dossiers,
- création, démarrage et arrêts de service,
- arrêt brutal de processus,
- changement des droits sur un fichier ou un dossier,
- installation de nouveaux composants…

## C. Création d'un serveur applicatif

Pour déployer une application web, on utilise un fichier Web Archive (war) [SILENCE13]. Ce fichier est composé des classes Java à exécuter, des pages web, des ressources associées (feuilles de style, JavaScript, images, …) et des servlets permettant de gérer les appels web et de les rediriger vers les méthodes Java associées. Ces fichiers ne peuvent être déployés que sur des serveurs applicatifs capables d'exécuter du Java.

### 1. Mise en place du serveur

Parmi les serveurs Open Source qui répondent aux exigences, trois grands noms ressortent : JBoss, Tomcat et GlassFish. JBoss vient de sortir une toute nouvelle version nommée WildFly [WildFly14], avec des performances optimisées. Contrairement à GlassFish, il intègre la gestion des websockets (cf. chap. III-A-1-d), indispensables au projet. Tomcat est un peu à la traine dans l'intégration des dernières fonctionnalités Java [MAPLE13].

L'installation du serveur est aisée. L'archive à télécharger fait 150Mo, une fois décompressée, il est possible de démarrer le serveur immédiatement. Une configuration par défaut, nommée « standalone », réunit les modules indispensables pour faire tourner le serveur (cf. annexe J.). L'arborescence est intuitive, dans le dossier « standalone » se trouvent les dossiers importants pour l'application :

- Dossier « configuration » : Contient les paramétrages du serveur dans le fichier « standalone.xml ». Ce fichier est structuré par module, les parties importantes pour FORMID sont le module de définition des traces, de définition de la base de données et du paramétrage des ports (cf. Annexe J). Dans le dossier configuration se trouvent aussi des fichiers de paramétrages système générés lors du démarrage du serveur.

  Ce dossier est visible par le code de l'application via la commande Java :

```
System.getProperties().get("jboss.server.config.dir");
```

*Extrait de code 3 : Commande pour trouver le dossier de configuration du serveur*

  Les fichiers de paramétrage de FORMID sont donc placés dans ce répertoire.

- Dossier « data » : Contient des dossiers système créés lors des déploiements d'applications. Ce dossier est également accessible depuis l'application grâce à la commande Java :

```
System.getProperties().get("jboss.server.data.dir");
```

*Extrait de code 4 : Commande pour trouver le dossier de données du serveur*

Les fichiers de données nécessaires à FORMID sont dans ce répertoire. On trouve notamment le code des simulations (jar, swf, fichiers d'initialisation, …) et les scénarios (fichiers descriptifs xml et code compilé Java).

- Dossier « log » : contient les fichiers de traces système : server.log et formid.log.
- Dossier « deployments » : surveillé par le serveur, dès qu'un fichier war y est déposé, il est automatiquement déployé.

### 2. Problèmes rencontrées

- Pour automatiser la mise en production de FORMID, l'outil Maven [Maven14] a été mis en place. Celui-ci télécharge les librairies manquantes nécessaires à la compilation, compile le code, génère la Javadoc, produit l'archive war et la dépose sur le serveur local.

  Certaines librairies placées dans l'archive par Maven existent déjà sur le serveur WildFly, et entrent donc en conflit lorsque celui-ci doit choisir entre l'une ou l'autre des versions. Pour solutionner ce problème, il faut indiquer dans la configuration Maven que la librairie est nécessaire uniquement à la compilation, et ne doit pas accompagner le code compilé sur le serveur. Par exemple pour la librairie RestEasy :

```
<dependency>
      <groupId>org.jboss.resteasy</groupId>
      <artifactId>resteasy-multipart-provider</artifactId>
      <version>3.0.4.Final</version>
      <scope>provided</scope>
</dependency>
```

*Extrait de code 5 : Exemple de configuration Maven pour éviter les conflits de librairies*

- Pour faciliter le travail des développeurs, des outils de débogage sont fournis dans les IDE (éditeurs de code), ils permettent d'exécuter des traitements en mode « pas à pas », c'est-à-dire instruction par instruction, avec une grande visibilité sur les variables manipulées. Les serveurs classiques sont intégrés aux IDE, ils peuvent être démarrés directement dans l'interface de travail. WildFly étant trop récent, les adaptateurs pour l'utiliser directement dans l'IDE n'existent pas encore.

  Pour contourner ce problème, il est possible d'effectuer un réglage sur le serveur, dans le fichier « standalone.bat » qui sert à lancer le serveur sur Windows.

```
set DEBUG_MODE=true
set DEBUG_PORT=8787
```

*Extrait de code 6 : Réglages pour démarrer le débogueur*

  Au démarrage du serveur, un port est ainsi dédié à la connexion vers un débogueur externe. Dans l'IDE, on peut dès lors lancer un débogueur distant pour travailler en mode « pas à pas » (cf. Figure 17).

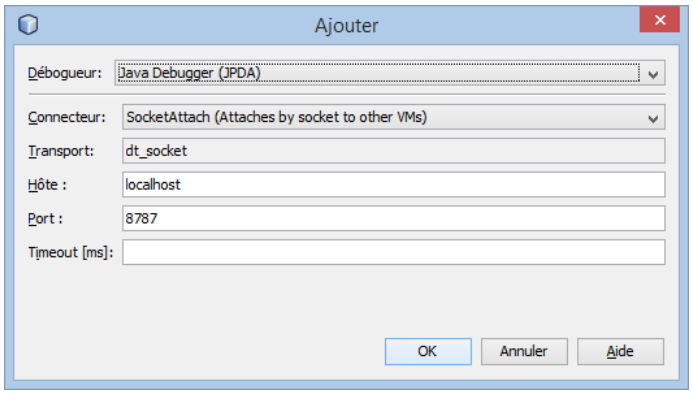



*Figure 17 : Ouverture d'un débogueur distant dans l'IDE*

- Dans l'optique de respecter les contraintes réseaux des établissements, le serveur avait initialement été configuré pour écouter le port 80. Lors de l'installation sur la machine virtuelle, la liaison ne se faisait plus. Il s'avère que le port 80 sur Linux, comme tous les autres en dessous de 1024, nécessite des droits d'administrateur pour être utilisé [PESTEL09]. Or par raison de sécurité, le serveur WildFly n'est pas lancé par un utilisateur ayant ces privilèges. Pour répondre à ce besoin, un serveur Apache a été mis en place, avec pour unique but de recevoir les appels sur le port 80 et les rediriger vers le port 8080 et donc WildFly.

```
<VirtualHost *:80 >
    ServerName formid.imag.fr

    ProxyRequests Off
    ProxyPass /ws/ ws://127.0.0.1:8080/ws/
    ProxyPassReverse /ws/ ws://127.0.0.1:8080/ws/

    ProxyPass / http://127.0.0.1:8080/
    ProxyPassReverse / http://127.0.0.1:8080/
</VirtualHost>
```

*Extrait de code 7 : Règles de redirection de ports sur le serveur Apache*

### 3. Base de données intégrée

Le serveur WildFly est livré avec une base de données relationnelle nommée H2Database. Celle-ci a été conservée pour l'application. Elle a l'avantage d'être intégrée au serveur, déjà installée, ce qui nécessite beaucoup moins de paramétrages, notamment pour la configuration des ports. Les performances de cette base sont suffisantes pour FORMID [H2 Database14]. Elle est livrée avec une interface web minimaliste (accessible à l'URL *http://localhost:8082*), qui permet d'exécuter des requêtes SQL et de parcourir les tables et leurs données. Cependant, pour un utilisateur ne maitrisant pas le langage SQL, il sera plus aisé d'utiliser un IDE avec des assistants de création ou d'interrogation de tables.

La base de données peut être lancée avec deux modes de connexion. Le premier embarque le moteur SQL dans la même JVM que l'application utilisant le connecteur JDBC (« Java DataBase Connectivity », interface entre une base de données et une application Java), dans ce cas-ci le serveur WildFly. C'est le mode le plus performant, mais cela ne permet

qu'une seule connexion ouverte en même temps. Cela n'est pas gênant pour une application en production, mais lors des développements, il n'est pas rare de tester l'application en surveillant l'apparition des données dans la base. Dans ce cas-là, il y a plusieurs connexions ouvertes, il faut donc choisir le deuxième mode, qui place le moteur SQL en tant que serveur accessible via le protocole TCP. La différence entre les deux configurations est minime pour le développeur, il s'agit de modifier l'URL de connexion, le moteur H2Database se charge du reste.

| | |
|---|---|
| **Mode embarqué** | *jdbc:h2:~/test* |
| **Mode serveur TCP** | *jdbc:h2:tcp://localhost/~/test* |

*Tableau 2 : URL de connexion à la base de données*

L'inconvénient principal de la base de données embarquée, c'est que le fichier de données s'enregistre au niveau de l'exécutable H2Database, et que celui-ci n'est pas facilement accessible dans la hiérarchie des répertoires du serveur.

```
[serveur] \modules\system\layers\base\com\h2database\h2\main
```

*Extrait de code 8 : Chemin d'accès au fichier physique de base de données*

Pour éviter de devoir parcourir cette arborescence, un exécutable placé à la racine du serveur a été prévu sous Windows pour ouvrir une nouvelle fenêtre directement au niveau de la base, et un dossier virtuel pointant sur la base a été créé sous Linux.

Actuellement, la modification de la base de données sur la machine virtuelle nécessite un rapatriement du fichier en local, l'exécution des scripts de mise à jour, et la livraison du fichier modifié. Une évolution intéressante serait de mettre en place une procédure de mise à jour de la base directement sur le serveur.

## D. Création de l'IHM

Le premier rapport entre un utilisateur et une application est lié à l'interface graphique. Un client remarquera d'abord un défaut d'alignement de boutons avant de repérer une incohérence de données. Même si la gravité de ces problèmes n'a pas du tout le même poids, il est important que le premier ressenti de l'utilisateur soit positif.

Pour rafraichir FORMID, l'IHM a été revue avec les librairies en vogue actuellement sur le web, ce qui permet de donner une impression de modernité, et une interface intuitive car en phase avec les sites les plus populaires du moment.

### 1. Librairie AngularJS

AngularJS [AngularJS14] est un framework JavaScript Open Source sponsorisé par Google. Il permet de créer des applications web de type « monopage » (SPA, « Single Page Application »), c'est-à-dire qu'il n'y a pas de changement d'URL durant la navigation, ni de rechargement complet de page, le JavaScript se charge de rafraichir les parties d'écran impactées par des actions de l'utilisateur.

AngularJS introduit une notion de Modèle-Vue-Contrôleur (MVC) du côté client, avec des liaisons de données dynamiques et bidirectionnelles entre la page HTML (la vue) et les données chargées dans le JavaScript (le modèle). Les actions de l'utilisateur sont gérées par le code JavaScript (le contrôleur) et peuvent modifier le modèle sans nécessairement faire un appel au serveur.

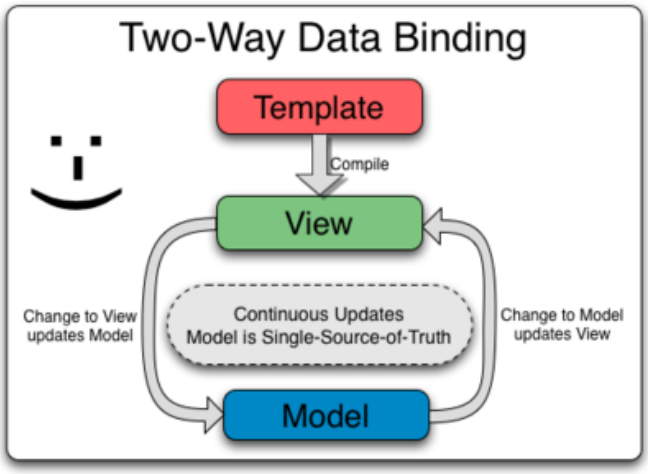



*Figure 18 : Liaison de données bidirectionnelle*

Une particularité intéressante fournie par AngularJS est la possibilité de créer de nouveaux éléments HTML. Ceux-ci seront compilés par le JavaScript pour être correctement interprétés à l'affichage (cf. annexe G.). La librairie en fournit déjà une collection, préfixée par « ng- ». Ils introduisent soit de nouveaux comportements, soit une amélioration des comportements existants. Par exemple, «  » sera traduit par une liste de balises « div », une pour chaque élément de « maCollection », ensemble de données présent dans mon modèle JavaScript.

De nombreux autres frameworks JavaScript ont vu le jour ces dernières années, certains fournissant les mêmes possibilités qu'AngularJS [PORTO13] . Il a fallu en choisir un, le tutoriel d'AngularJS était plutôt bien réalisé, et la librairie, plus ancienne, semblait suffisamment mature pour ne pas avoir trop de défauts de jeunesse.

### 2. Librairie Bootstrap

La librairie Bootstrap [Bootstrap14] est essentiellement graphique. Elle a été initialement développée par la société Twitter avant de passer sous licence Open Source en 2011. Elle comprend une série d'outils utiles à la création de sites web. Elle contient une feuille de style CSS (« Cascading Style Sheet ») qui définit l'apparence de tous les éléments HTML, ce qui uniformise l'ensemble du site web. Elle connait un grand succès depuis 2012, beaucoup de grandes entreprises l'utilisent pour leur site [Love Bootstrap14].

Elle présente deux gros avantages :

- Pour le développeur, il obtient très vite un résultat présentable, harmonieux, il ne doit pas jouer le rôle du graphiste, la présentation et les couleurs sont standards.
- Pour l'utilisateur, il retrouve des éléments connus, la forme des menus ou des boutons, la couleur, la disposition de l'écran, cela lui permet une meilleure prise en main.

### 3. Librairies utilitaires

Actuellement, il existe des milliers de librairies JavaScript, fournissant des services de plus ou moins grande importance, et qui permettent de ne pas devoir redévelopper des fonctionnalités ou des composants existants ailleurs. Il faut éviter d'en abuser, la prise en main peut être plus coûteuse que le développement personnel, les défauts ne sont pas toujours visibles et les fonctionnalités peuvent impacter d'autres librairies et provoquer des comportements inappropriés. Cependant, certaines librairies sont indispensables et le gain de temps est vraiment significatif.

#### *a. JQuery*

JQuery [JQuery14] est une librairie JavaScript gratuite, largement répandue sur le web [TIT_TOINOU13]. Elle fournit un ensemble d'outils simples d'utilisation pour manipuler le DOM (« Document Object Model »), c'est-à-dire la structure en arbre de la page HTML. JQuery permet ainsi la sélection d'éléments dans la page HTML, la modification de contenu, et l'ajout de nouveaux éléments. Elle permet aussi la gestion d'évènements et facilite les appels au serveur via la technologie AJAX (« Asynchronous JavaScript And XML »).

JQuery est compatible avec tous les navigateurs existants et n'impacte à priori pas les autres librairies JavaScript, au contraire, elle est souvent pré-requise pour faire fonctionner d'autres frameworks tel que AngularJS. Il existe des modules complémentaires pour enrichir les fonctionnalités de JQuery, comme « JQuery UI » qui fournit des composants graphiques (sélecteur de date, menu tabulaire, barre de progression, …).

#### *b. Restangular*

Restangular [GONTOVNIKAS13] est une librairie JavaScript enrichissant AngularJS, pour faciliter la liaison au serveur lorsque les appels sont de type REST (cf. chap. III-A-1-c). L'intérêt est de pouvoir créer une requête sans devoir reconstruire les différents entêtes, en factorisant les parties communes de l'URL, et en gérant la réponse du serveur.

La librairie fournit toutes les méthodes nécessaires pour aller chercher un élément ou une liste d'éléments, pour envoyer des données ou pour construire une URL complète. Lors de

l'initialisation de la librairie, on lui donne le préfixe commun des appels au serveur, les champs à passer dans l'en-tête de la requête tel que le jeton d'authentification, et les traitements génériques à réaliser après chaque retour du serveur.

| Appel classique | Appel Restangular |
|---|---|
| `var myURL = prefixe + « /user/ » +userId+ « /projects » ;`<br>`$http.get(myURL).`<br>`success(function(data) {`<br>`//traitement de ma liste`<br>`});` | `Restangular.one("user" ,userId)`<br>`.all("projects")`<br>`.getList().then(function(data)`<br>`{`<br>`//traitement de ma liste`<br>`}) ;` |

*Tableau 3 : Différence d'appels avec ou sans la librairie Restangular*

# III. FORMID – Modification de l'existant

Le choix de refonte de l'architecture a entrainé des modifications majeures de la suite logicielle FORMID. Tout d'abord la migration des applets Java vers un mode client-serveur a demandé une réflexion sur le mode de communication et sur la séparation entre ce qui s'exécuterait sur le client et ce qui serait à charge du serveur. Ensuite l'abandon du serveur testbed a nécessité le développement d'une couche de gestion des utilisateurs, avec l'authentification associée et la gestion des sessions. Finalement l'évolution de l'IHM a levé des questions d'ergonomie, la disposition des différents écrans a été revue pour qu'elle soit plus intuitive et plus fonctionnelle.

## A. Serveur applicatif

Le serveur applicatif est le programme capable d'exécuter FORMID, de paramétrer la gestion des utilisateurs et de leurs privilèges, et de fournir des écrans d'administration pour la gestion des apprenants, professeurs et séances de cours.

### 1. Choix techniques

Les technologies utilisées ont été sélectionnées en utilisant des critères de maintenabilité, facilité de mise en œuvre, de performances, et de modernité. L'objectif est que le code ait une pérennité maximale, et que les futurs intervenants, même s'ils n'ont pas un niveau avancé en programmation, n'aient pas de difficultés à comprendre les techniques employées.

#### *a. Java*

Le Java est un langage de programmation orienté objet, Open Source, utilisé partout dans le monde, dans de nombreux domaines et notamment dans les milieux académiques. Il est très souvent enseigné dans les différents cursus informatiques. La suite logicielle FORMID ayant été conçue en Java, ce choix n'a pas été remis en cause. Cependant, le framework utilisé datant de 2006 (Java 1.2 – Java 1.4), il a été remplacé par la version 1.7 datant de 2011 et qui est encore régulièrement mise à jour par Oracle. Certains composants ont dû être migrés ou remplacés, les outils du langage ayant évolué.

Autant que possible, le code a été réutilisé. Cela concerne essentiellement la gestion des scénarios, la génération, la compilation, le chargement dynamique des classes compilées. Certaines classes ont gardé leur logique mais ont évolué avec des composants Java plus performants, comme la lecture ou l'écriture de fichiers XML. La partie événementielle très intégrée à l'interface, la gestion des traces ou la succession des exercices a complètement été recodée.

#### *b. JPA*

JPA (« Java Persistance API ») est un module Java permettant d'accéder à une base de données à partir du code Java, en nécessitant très peu de programmation. Ce module est composé de trois parties.

- Tout d'abord des annotations Java. Les annotations sont des mots clés préfixés d'un « @ », qui sont placés soit au-dessus du nom de la classe, soit au-dessus d'une

méthode ou d'un attribut de la classe. Ces annotations sont traduites par le compilateur qui génère du code à la place du développeur. Les annotations JPA permettent la génération du code pour accéder aux données. Chaque classe représente un objet (donc une table) de la base de données. Les annotations permettent de définir à quelle table attacher la classe et de lier les attributs de la classe aux colonnes associées.

```
@Entity
@Table(name = "COURSES")
@XmlRootElement
public class Course implements Serializable {

    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "NAME")
    private String name;
```

*Extrait de code 9 : Exemple d'utilisation des annotations JPA*

- Ensuite un fichier « persistance unit », nommé « persistence.xml », décrit d'une part les classes Java représentant des tables de la base, et d'autre part le connecteur JDBC à utiliser pour communiquer avec le type de base utilisé, dans notre cas « H2Database ». Ce fichier peut facilement être généré par l'IDE, il sera mis à jour à chaque nouvelle création de classe utilisant JPA.

- Enfin, une interface permettant de créer des requêtes plus complexes sur la base, avec des critères de recherche ou des liaisons de table. Le développeur ne doit de cette manière jamais écrire de SQL, et les requêtes ne sont pas liées à un moteur SQL en particulier. Les requêtes peuvent être écrites soit sous forme de chaine de caractères, soit en utilisant un constructeur de requêtes fourni par JPA.

| Requête sous forme de chaine de caractères | Requête via le constructeur JPA |
|---|---|
| `SELECT u FROM User u WHERE UPPER(u.login) = :login`<br><br>`//User = nom de classe Java`<br>`//login = attribut de la classe Java` | `CriteriaQuery cq = getEntityManager()`<br>`  .getCriteriaBuilder().createQuery();`<br>`cq.select(cq.from(User.class));`<br>`Query q = getEntityManager()`<br>`  .createQuery(cq);`<br>`q.setMaxResults(20);`<br><br>`List<User> users = q.getResultList();` |

*Tableau 4 : Exemple de requêtes construites avec JPA*

Grâce à JPA, la persistance des objets est donc simplifiée, le développeur ne doit pas se soucier des conversions lorsqu'il crée et récupère un objet dans la base de données. S'il est à l'aise avec les requêtes SQL, il peut écrire ses requêtes à la main, s'il veut être sûr de ne pas faire d'erreur lors de requêtes complexes avec liaison de tables et filtres évolués, il vaut mieux utiliser le constructeur JPA.

### c. REST

REST (« REpresentational State Transfert ») est un type d'architecture de service web, permettant la représentation de la logique de l'internet [BORDERIE03]. Il définit les bonnes pratiques pour créer un service web en adressant chacune des ressources par une URL unique. Les paramètres que l'on a l'habitude de placer en annexe de l'URL (derrière un « ? ») deviennent partie intégrante de l'adresse : www.server.test?userId=1 devient www.server.test/userId/1. L'utilisateur 1, en tant que ressource, a sa propre URL. Cela permet de n'utiliser que le protocole « http » pour appeler un service, là où d'autres types de webservices doivent ajouter des méthodes et donc enrichir le protocole pour obtenir le même résultat.

Java a introduit dans sa norme une standardisation de l'implémentation de REST. La librairie utilisée dans FORMID est RESTeasy [RESTEasy14], car elle est embarquée par défaut par le serveur WildFly. Pour l'utiliser, il faut déclarer une servlet (point d'entrée des appels au serveur pour Java) dans le web.xml.

```
<servlet id="restWS">
    <servlet-name>FormID REST Service</servlet-name>
    <servlet-class>org.jboss.resteasy.plugins.
server.servlet.HttpServletDispatcher</servlet-class>
    <init-param>
      <param-name>javax.ws.rs.Application</param-name>
      <param-value>fr.imag.lig.metah.
formidweb.rest.service.ApplicationConfig</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>FormID REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
```

*Extrait de code 10 : Déclaration d'un webservice REST dans le web.xml*

De cette manière, tous les appels correspondant au modèle défini, c'est-à-dire dans notre cas préfixés par « /rest/ », seront redirigés vers le gestionnaire Java de services REST.

```
http://formid.imag.fr/rest/users/students/0/10/id/asc
=>
http://nomDuServeur/rest/classeUsers/méthodeStudents/paramètres
```

*Extrait de code 11 : Exemple d'appel REST*

Il faut ensuite créer des classes Java capables de comprendre les appels et d'y répondre. On utilise pour cela les annotations fournies par RESTEasy. Pour déclarer une classe capable de répondre à un appel REST, on emploie l'annotation « *@Provider* », et on lui affecte un chemin via l'annotation « @Path("monChemin") », pour permettre au gestionnaire d'affecter correctement les requêtes aux méthodes associées. Ensuite on implémente des méthodes qui rendent les services voulus d'interrogation, de création, de modification ou de suppression de données, avec pour chaque méthode un type d'appel standard web (GET, POST, PUT et DELETE), une spécialisation du chemin et éventuellement le type de

données accepté en entrée ou en sortie. Le chemin peut également contenir des paramètres qui seront récupérés par la méthode.

```
@Provider
@Path("users")
public class UsersRESTFacade {

    @POST
    @Consumes("application/json")
    public Response create(UserDetail _user){
        [...]
    }

    @GET
    @Path("{first}/{max}")
    @Produces("application/json")
    public List<UserDetail> findFilteredUsers(
                @PathParam("first") Integer first,
                @PathParam("max") Integer max) {
        [...]
    }
}
```

*Extrait de code 12 : Exemple d'utilisation d'annotations RESTEasy*

Le format classique d'échange de données est le JavaScript Object Notation (JSON), il est préféré au XML car il est plus simple et plus léger [SOUBOK06]. Chaque objet est délimité par des accolades, et est composé d'associations « clé-valeur », où « clé » est une chaine de caractères, et « valeur » soit un « type primitif » (chaine de caractères, nombre, booléen, …) soit un objet JSON, soit un tableau de « valeur », délimités par des crochets. Il peut être converti en objet JavaScript ou servir à reconstruire un objet Java.

```
{
  "id"        : 2,
  "name"      : "ALBERT",
  "firstname" : "Jean",
  "groupe":{
            "id"   :7,
            "name" : "classe 1"
        },
  "roles":[
            {
                  "id"    : 1,
                  "name"  : "ADMIN",
                  "privs" : "admin"
            },
            {
                  "id" : 2,
                  "name":"AUTHOR",
                  "privs":"author"
            }
      ]
}
```

*Extrait de code 13 : Exemple d'objet JSON*

### d. Websocket

Un des points importants de l'exécution du module élève de FORMID, est la réactivité du logiciel lorsque l'apprenant fait une action sur la simulation. Lorsqu'on fonctionne en applet Java, comme le code tourne sur la même page que la simulation, l'application peut interroger la simulation en permanence (en réalité toutes les 400 millisecondes) et analyser si l'état de l'application nécessite une réaction de FORMID et donc l'apparition d'un message dans la zone de notification de l'apprenant.

Pour garantir la réactivité de l'écran dans le mode client-serveur, il ne faut pas faire d'appel bloquant au serveur durant un exercice. Il ne faut donc pas attendre la réponse du serveur lors de l'envoi de l'état courant de la simulation. Mais il faut garantir que le serveur puisse envoyer des messages à l'utilisateur. La piste explorée est la mise en place de websockets.

Les websockets sont un nouveau standard web, donc la norme d'implémentation a été introduite dans Java 7 [HADJIAT11]. Ils permettent la communication bidirectionnelle permanente entre un client et un serveur. Cela permet au serveur de pousser des données vers le client sans appel de celui-ci. Il peut donc communiquer ses changements d'état ou réagir à des évènements extérieurs et en informer les clients connectés.

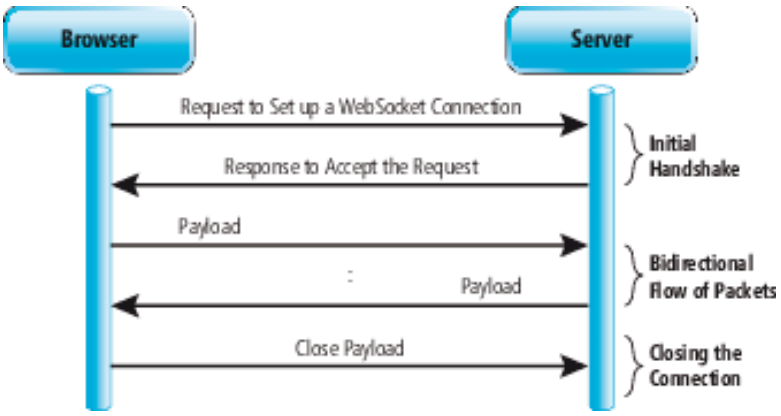



*Figure 19 : Mode de fonctionnement du protocole Websocket*

Pour FORMID, cela signifie que les appels sont dissociés, le client avertira des changements d'état de la simulation sans attendre de réponse, le serveur statuera sur la présence ou non d'une situation remarquable telle que définie dans le scénario, et au besoin enverra le message correspondant au client.

Lors du démarrage d'un exercice par un apprenant, le client demandera un identifiant de canal pour communiquer via websocket. Cet identifiant est une chaine de caractères aléatoire qui est réservée pour l'apprenant. Une fois cet identifiant enregistré dans le contexte JavaScript, le client va s'en servir pour demander une connexion websocket, le serveur associe alors la chaine de caractères du canal au client, il sait dès lors que tout message reçu sur ce canal provient de ce client, et que, lorsque le serveur doit envoyer une information à cet apprenant, il peut utiliser ce canal.

Un souci majeur de la solution mise en place est la compatibilité avec les anciens navigateurs[1]. Pour simuler le fonctionnement des websockets sur les navigateurs qui ne supportent pas ce protocole, on utilise la méthode de « long-pooling ». Il s'agit de lancer une requête inutile au serveur, celui-ci la bloque le temps d'avoir quelque chose à dire, et s'en sert comme canal de communication au moment opportun. Si c'est le client qui a quelque chose à dire, il annule sa requête et en lance une nouvelle avec son message.

Une librairie nommée Atmosphere [Atmosphere Framework14] a été mise en œuvre pour gérer ce mécanisme, elle travaille aussi bien avec les websockets qu'avec le « long-pooling », de manière transparente pour le développeur. Elle est composée de deux modules, un pour le côté serveur en Java, l'autre pour le client en JavaScript (cf. annexe F.).

Pour le Java, cela fonctionne par annotation, la classe de gestion de websocket a une annotation « @ManagedService(path = "{studentws: [a-zA-Z][a-zA-Z_0-9]*}") » qui définit un chemin pour spécialiser l'URL. Ensuite trois méthodes doivent être implémentées, avec les annotations « @Ready », « @Disconnect » et « @Message ». La première méthode initialise la connexion, la seconde y met fin et la dernière traite les messages qui arrivent. Une méthode a été rajoutée pour envoyer un message sur un canal donné.

Du côté JavaScript, la librairie permet de s'inscrire sur un canal, d'y pousser des messages et de lire les messages entrants. De plus, elle est capable de gérer le type de connexion, si le client ne parvient pas à créer un websocket, la librairie bascule en mode « long-pooling ».

### 2. Services utilitaires

Pour pouvoir rendre les services qui étaient auparavant à la charge du serveur testbed, il faut fournir une gestion des utilisateurs et de leurs privilèges, garantir la sécurité des données, et permettre à des administrateurs de créer les classes d'élèves, de leur attribuer un professeur, et de créer des séances de cours composées de scénarios pédagogiques.

#### *a. Gestion de la sécurité*

Plusieurs systèmes de sécurité ont été placés en parallèle. Le premier, le plus simple et le plus courant, est une connexion par nom d'utilisateur et mot de passe. Un jeton d'authentification est généré et placé dans l'en-tête de la requête. Ce jeton est généré en suivant un modèle « nom d'utilisateur : date d'expiration du jeton en milliseconde : signature ». La signature est composée du nom d'utilisateur, de son mot de passe, de la date d'expiration du jeton et d'un mot-clé. Elle est ensuite cryptée grâce à l'algorithme standard MD5 et encodée en hexadécimal pour une sécurité accrue.

Chaque utilisateur a une liste de rôles associés, qui interviennent à deux niveaux :

- Au niveau serveur, chaque requête REST est interceptée. Le jeton d'authentification est extrait de l'en-tête et sa validité est testée. Si c'est le cas et que la session n'est pas expirée, on récupère les rôles associés à l'utilisateur. On inspecte la méthode appelée, celle-ci peut contenir plusieurs annotations fournies par la librairie RESTEasy : « @DenyAll », cette méthode n'est pas accessible, « @PermitAll », cette méthode est publique, ou « @RolesAllowed({"ROLE1", "ROLE2"}), seuls les

[1] Actuellement, Internet Exporer 9 et inférieurs. Chrome, Mozilla Firefox et IE10 sont compatibles.

utilisateurs possédant un des rôles listés ont accès à la méthode. Un utilisateur a le droit d'avoir plusieurs rôles, ce qui permet de faire une gestion très fine des accès aux méthodes, certaines n'étant accessibles qu'aux apprenants et pas aux administrateurs, lesquels pouvant par contre s'attribuer le rôle « apprenant » s'ils le désirent.

- Au niveau de l'interface, l'utilisateur, son jeton d'authentification et ses rôles sont stockés dans une variable de session. Cette variable possède elle aussi un délai d'expiration, pour éviter de pouvoir parcourir les écrans sans appel au serveur après déconnexion. Chaque action dans l'écran rafraichit ce délai. Lors de la connexion, un calcul de privilège est réalisé sur base des rôles de l'utilisateur. On se sert pour cela d'un masque de bits. Chaque rôle a une position dans un nombre binaire, les rôles de l'utilisateur ayant leur bit positionné à 1, les rôles que l'utilisateur ne possède pas restant à 0. Chaque onglet du menu a aussi son masque binaire. De cette manière, il suffit de faire un « et binaire » pour valider que l'utilisateur a accès à l'écran (cf. annexe H.).

```
    010 // = rôle utilisateur: apprenant
AND 100 // = privilège écran : administrateur
=   000 // = 0, pas d'accès

    010 // = rôle utilisateur: apprenant
AND 111 // = privilège écran : public
=   010 // = Valeur différente de 0, accès autorisé
```

*Extrait de code 14 : Exemple de calcul binaire pour les privilèges*

### b. Écrans de paramétrage

La configuration des groupes, élèves, tuteurs, séances de cours et exercices proposés durant celles-ci était réalisée via des fichiers texte posés sur le serveur. Cela permettait de composer facilement une classe, mais présentait plusieurs inconvénients, il n'y avait pas d'historique des fichiers, il fallait un accès serveur à chaque changement de configuration, et il n'y avait pas de lien entre un tuteur et un groupe.

Des écrans de paramétrage des différents éléments stockés en base de données ont été réalisés (cf. annexe C.). Ils permettent à l'administrateur de créer ou éditer des utilisateurs, de leur attribuer des rôles, de les lier à des groupes en tant qu'élève ou que tuteur. L'attachement peut aussi se faire dans la création ou l'édition d'un groupe.

La création de cours est complexe, elle permet d'y attacher des exercices liés à des scénarios, d'en décider l'ordre. Elle permet de créer des séances de cours et de les attribuer à un groupe. Il est possible également de dupliquer une séance existante en changeant le groupe et l'heure de la séance.

Un écran d'opérations système permet de réaliser des actions à priori plus sensibles et moins fréquentes. On y trouve l'export de traces, la compilation directe d'un scénario au format XML, l'enregistrement d'une nouvelle simulation et l'édition des paramètres pour l'accès à un serveur d'ontologie.

Pour l'enregistrement d'une nouvelle simulation, cela doit être accompagné d'un dépôt sur le serveur du fichier faisant tourner la simulation (swf ou jar), et de la création d'un fichier HTML appelant ce fichier, avec l'implémentation en JavaScript des méthodes de dialogue avec FORMID (« getVariables », « setVariables », « getVariablesNames », « getVariablesValues » et « getVariablesTypes »). Si la simulation ne permet pas de répondre à ces appels, elle ne sera pas compatible avec FORMID.

## B. Module élève

Le module élève est l'interface accessible aux apprenants. Il permet de choisir un exercice, de travailler dessus, de recevoir des consignes pour chacune des étapes composant l'exercice, d'avoir des messages en fonction des actions réalisées sur la simulation associée, et de demander des validations de fin d'étape.

### 1. Changement de l'interface

La disposition de l'écran n'a pas été remaniée (cf. annexe C.). Le choix d'exercices se fait dans la partie supérieure de l'écran, la simulation s'affiche dans la partie gauche et la zone FORMID est à droite. La taille a été optimisée. En effet, les applets Java avaient une taille fixée dans le code de l'IHM, non modifiable, et utilisée dans la page HTML. Les zones de texte notamment étaient trop petites, et les messages n'étaient pas évidents à lire. L'écran est dorénavant divisé en deux parts égales, une moitié pour la simulation qui pourra prendre toute la place disponible (en fonction de l'écran de l'utilisateur), et l'autre moitié pour FORMID.

### 2. Messages de suivi

Les messages de suivi ont pu être améliorés de manière simple. Leur apparition n'était pas assez visible dans l'ancienne interface, et il était difficile de faire la distinction entre les différents types de message (consignes, erreurs, …). Pour contourner le problème, les auteurs de scénarios inséraient des indentations et des caractères spéciaux pour différencier les messages.

La zone de message a été séparée en deux, une première partie (cf. Figure 20) affiche les consignes, en caractères gras pour les messages d'exercice et en italique pour les consignes d'étape.

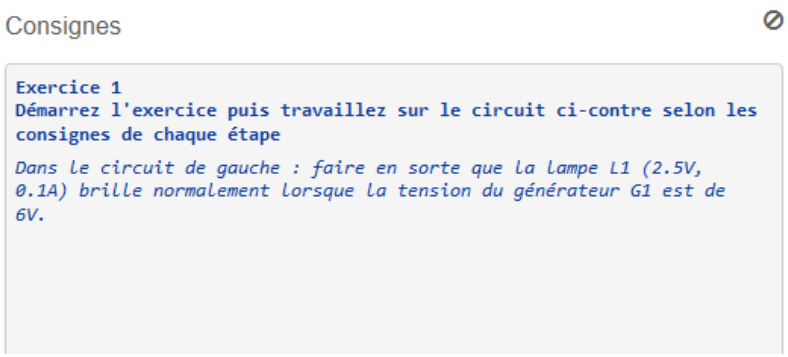



*Figure 20 : Zone de consigne d'exercice ou d'étape*

La deuxième zone (cf. Figure 21) affiche les messages liés aux validations d'exercices et messages de contrôles provoqués par un état de la simulation déclaré comme remarquable dans le scénario. Les messages d'erreur sont en rouge, ceux délivrés en cas de réussite sont en vert.

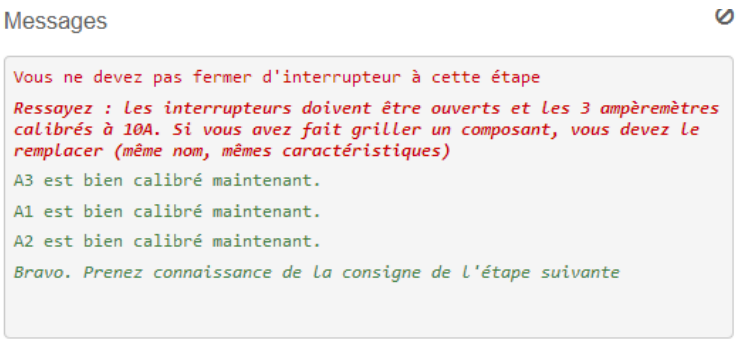



*Figure 21 : Zone de message concernant l'avancement du scénario*

De cette manière, grâce à la séparation entre consignes et messages et à l'utilisation de codes couleur, les instructions sont plus parlantes pour l'apprenant.

### 3. Interactions avec la simulation

Les interactions avec la simulation se font via JavaScript. Au démarrage de l'exercice, un minuteur est mis en place, pour faire un appel à la simulation. Les variables déclarées dans le scénario sont pré-chargées dans le contexte JavaScript, ce qui permet d'interroger la simulation uniquement sur l'état des variables pertinentes pour le scénario. On n'envoie l'état de la simulation au serveur que si au moins une variable a changé de valeur, ce qui évite des appels inutiles au serveur.

Lorsque le serveur reçoit une notification de changement d'état de la simulation, il met à jour l'état interne de l'avancement de l'élève, vérifie que ce changement ne déclenche pas un des contrôles déclarés pour le scénario en cours, et si un message doit être envoyé, le serveur utilise le canal dédié à l'apprenant.

Si l'élève fait une demande de validation, le serveur se sert du dernier état enregistré dans l'avancement pour tester la réussite de l'étape, et envoie le résultat au client.

### 4. Traces des événements

Les traces des événements ont été adaptées à la nouvelle architecture. Auparavant, elles étaient représentées sous la forme d'une chaine de caractères normée : « idSeance#date#idEleve#idContenu#numExo#numEtape#numControle#score#contenu »

Elles étaient enregistrées telles quelles dans une base de données. Une nouvelle modélisation a été mise en place (cf. Figure 22), en extrayant les objets Séance, Utilisateur et Exercice, et en plaçant en colonne les éléments restants.

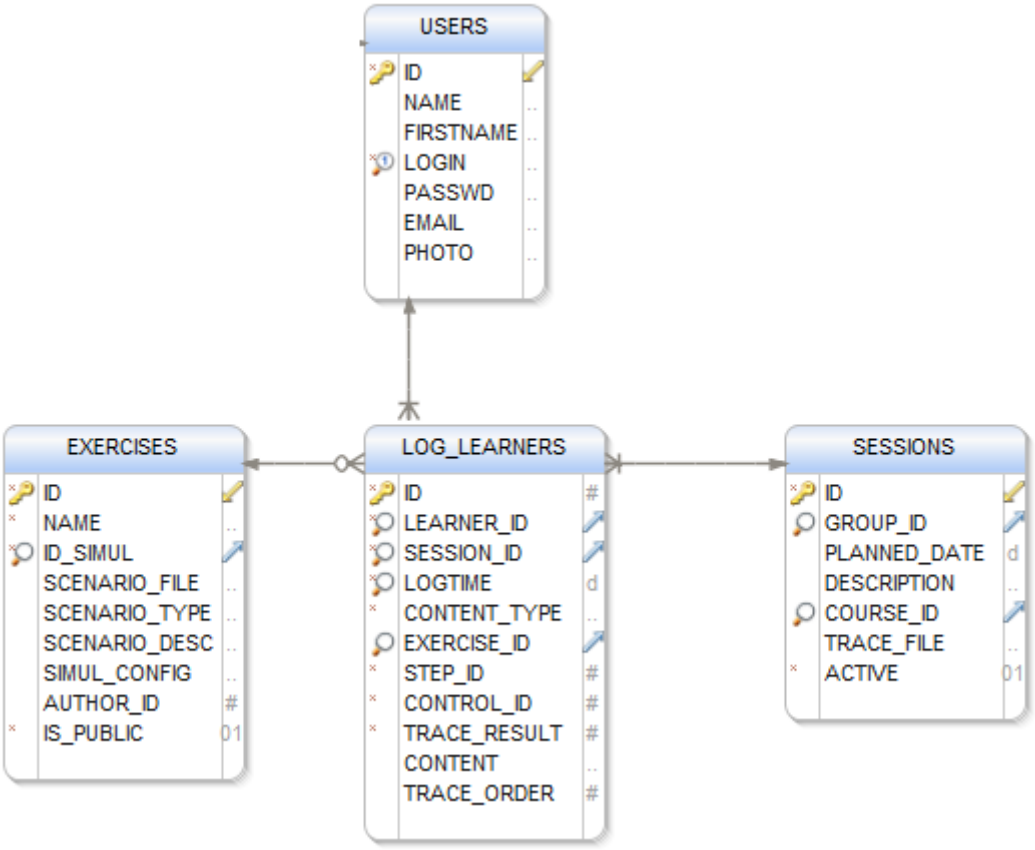



*Figure 22 : Schéma de base de données - Vue "traces élève"*

Un travail de migration des traces existantes a été effectué. Un mécanisme de création des utilisateurs, groupes, cours, séances de cours, et exercices a été réalisé pour pouvoir relier chaque trace à des éléments de la base de données. Ce déplacement vers la nouvelle modélisation était important, car les traces sont utilisées pour rejouer des séances dans le module tuteur.

## C. Module tuteur

Le module tuteur permet d'effectuer le suivi d'une classe, en temps réel ou de manière asynchrone. L'utilisateur choisit une classe et une séance de cours (instance d'un cours attribué à un groupe et à un horaire) et l'interface affiche l'avancement de la classe. Il peut rafraichir l'écran manuellement ou de manière automatique.

### 1. Changement de l'interface

De nouveau, la disposition de l'écran n'a pas beaucoup évolué (cf. annexe C.). L'espace alloué à l'affichage a été maximisé, afin d'agrandir les zones d'affichage d'information par survol. Le choix de changement de style d'interface (qui permettait de faire varier la couleur des boutons dans l'applet) a été supprimé, il n'avait pas de valeur ajoutée. Le mode « debug » a aussi été supprimé, les traces système se situant au niveau du serveur. Il n'était pas approprié que ce choix soit à la portée d'un utilisateur lambda.

Pour optimiser l'espace disponible, la possibilité de réduire les informations contextuelles de la séance a été introduite. Les fenêtres de type « pop-up » lors du survol des zones de suivi ont été redessinées, en permettant aux données qui le nécessitent de se structurer en tableau.

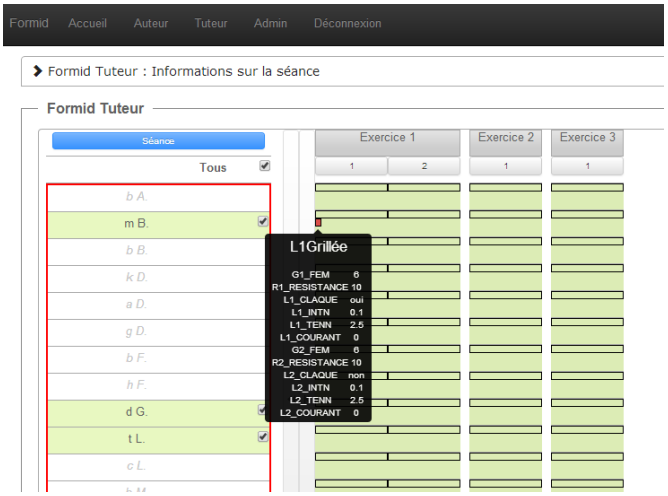



*Figure 23 : Fenêtre "pop-up" avec les données sous forme de tableau*

La sélection d'élèves a été améliorée. Auparavant, il fallait désélectionner tous les élèves pour pouvoir en choisir un sous-ensemble. Le bouton de sélection/désélection de l'ensemble de la classe était binaire. Un mode ternaire a été mis en place. Cela permet de pouvoir sélectionner ou désélectionner un élève indépendamment de l'état de la case à cocher « Tous les élèves ». De plus, il y avait un problème lors de la sélection unique d'élève. Lorsqu'un élève et une étape sont sélectionnés, un écran spécifique apparait avec la chronologie des actions de l'élève pour cette étape. Dans l'ancien modèle, pour voir le même écran pour un autre élève, il fallait décocher l'élève courant et cocher l'élève ciblé. Cela entrainait un changement de vue, la condition 1 élève/1 étape n'étant plus respectée. Pour rester sur le même écran, la possibilité d'utiliser le clic droit a été mise en œuvre. Le clic droit entraine la sélection unique de l'élève ciblé. La condition est donc respectée et l'écran ne bascule pas dans un autre mode. De plus, c'est un gain de temps lorsqu'on travaille sur un sous-groupe d'élèves et que l'on désire se focaliser sur un apprenant en particulier. Au lieu de décocher les autres élèves, un simple clic droit sur l'élève désiré suffit.

### 2. Récupération des traces élève

Deux modes ont été développés pour récupérer des traces élève. Le premier va lire un fichier texte ligne par ligne, l'autre interroge la base de données. C'est ce dernier qui est actif actuellement. Chaque trace en base a un numéro de séquence, auto-généré à la création. À chaque appel du client, ce numéro est rapatrié, et est passé en paramètre de l'appel suivant. La requête en base prend la trace de la même séance de cours, qui possède le plus petit numéro de séquence strictement supérieur à la séquence reçue en paramètre de la méthode. Si la requête ne renvoie rien, c'est qu'on est au bout de la séance, ou qu'on la suit en temps réel et qu'aucun apprenant n'a produit de trace depuis le dernier appel.

Toutes les traces récupérées sont placées dans un modèle de données JavaScript, et celui-ci est lié à l'écran affiché. Il est composé d'une liste d'élèves, chacun étant associé à une

liste d'exercices, eux-mêmes contenant une liste de contrôles. Suivant son type (validation, connexion, contrôle, …), une trace est placée à un niveau déterminé du modèle. Dès qu'une trace est insérée dans le modèle, l'écran est mis à jour automatiquement. Lorsqu'un changement de vue est effectué à la suite d'une sélection d'élève ou d'étape, l'écran se reconstruit avec le modèle de données et toutes les traces le composant.

Les traces de validation et de contrôle sont les plus nombreuses et les plus critiques, c'est en effet les plus parlantes pour suivre l'avancement d'un élève. Elles amènent dans l'interface des fonctionnalités de survol, de chronologie et se situent au croisement entre un élève donné et une étape ou un contrôle donné. À chaque nouvelle trace, du code HTML doit être inséré dans le tableau, lequel contient des liaisons de données AngularJS. Pour que le navigateur interprète correctement cet ajout, des directives complexes ont été développées (Voir exemple en annexe F). Elles reçoivent la trace, utilisent les données pour générer du code HTML sur base d'un modèle qui comprend des instructions AngularJS que le navigateur ne peut pas interpréter. Pour éviter que l'affichage soit incomplet, le code généré est compilé par le moteur AngularJS, les instructions AngularJS sont transformées en HTML compréhensible par le navigateur, puis le résultat est inséré dans la page.

### 3. Problème de rafraîchissement de l'écran

La liaison de données bidirectionnelle fournie par AngularJS est puissante, mais elle a ses limites. À chaque modification du modèle, tous les éléments de la vue attachés au modèle sont recalculés, même ceux qui ne sont pas impactés par la modification. AngularJS fixe une limite moyenne à 2000 liaisons sur une page, cela varie en fonction du navigateur et des performances de l'ordinateur du client [NICOLA13]. Cela parait beaucoup, mais sur une classe de 20 élèves, pour un cours composé de 15 exercices comprenant chacun 5 étapes, si chaque apprenant produit une trace par étape, il y aura déjà 1500 liaisons.

Or il faut savoir qu'une trace produit à elle seule trois liaisons, une pour la fenêtre « pop-up » qui donne le détail de la trace, une pour la couleur en fonction du résultat, et une dernière pour l'affichage (en effet, la trace ne sera pas visible si l'apprenant n'est pas sélectionné). D'autres éléments tels que le nom de l'élève ou la liste des exercices contiennent eux aussi des liaisons. Sur des petits groupes ou des séances avec peu d'exercices, on ne constate aucun ralentissement. Mais les tests sur des exemples plus conséquents n'ont pas du tout été concluants. Le temps d'affichage du premier écran était de plus d'une minute, et chaque nouvelle trace mettait plusieurs secondes à s'afficher.

L'idée a donc été de reprendre l'ancien modèle. Dans les applets Java, le nombre de colonnes affichées était fixe, et des boutons permettaient d'accéder aux exercices ou contrôles non visibles. Les ascenseurs choisis dans la nouvelle interface impliquaient que l'entièreté de l'écran soit calculée, même si des parties étaient non visibles. Le calcul a été restreint aux éléments visibles et des boutons ont été placés pour la navigation. Pour éviter que l'arrivée d'une nouvelle trace pour un exercice non visible impacte l'écran, un sous-ensemble du modèle a été extrait et lié à l'écran. Ce sous-ensemble est recalculé lors de la navigation. Les performances ont été grandement améliorées, avec un temps d'affichage de moins de 5 secondes, et l'apparition des nouvelles traces pratiquement en temps réel.

## D. Module auteur

Le module auteur permet la création de scénarios pédagogiques, en lien avec une simulation du domaine étudié. Le scénario est éditable, avec des automatismes qui vont chercher les données dans la simulation, et/ou à la main. Un scénario respecte une structure définie, qui peut être représentée par un arbre. Il peut être sauvegardé dans un fichier XML et être compilé en Java.

### 1. Changement de l'interface

L'interface a été complètement retravaillée (cf. annexe C.). Auparavant, la simulation était l'élément central, et l'arbre d'édition du scénario était dans un onglet, avec un deuxième onglet pour le choix de variables de la simulation à lier au scénario. L'auteur jonglait entre ces deux affichages, les actions liées aux variables se trouvant dans le panneau d'édition du scénario.

L'arbre du scénario, élément central du module, est désormais dans la partie gauche de l'écran qui lui est réservée, la simulation et l'écran de choix de ses variables sont dans la partie droite, dans des panneaux de type « accordéon vertical ». L'utilisateur peut les replier et déplier à sa guise en fonction des parties sur lesquelles il travaille. Les actions liées à l'édition du scénario restent accessibles au sommet de l'écran. Un menu déroulant réunit les fonctions liées à la gestion de fichier : nouveau scénario, sauvegarde, ouverture, compilation.

L'édition d'un élément du scénario se fait maintenant par double-clic sur le nœud correspondant dans l'arbre. Une fenêtre s'ouvre pour l'édition et le type d'élément à éditer influence la zone de saisie : simple champ pour une chaine de caractères, case à cocher pour une valeur booléenne, large zone éditable pour les équations de validation ou les états initiaux et finaux des étapes.

L'arbre est construit grâce à une librairie JavaScript nommée « jstree » [jsTree14]. Elle prend en entrée une hiérarchie de données composée d'éléments, de leur type et de leurs enfants. Lors de l'instanciation de l'arbre, on peut définir les types de nœud, et le type de leurs enfants éventuels. Des méthodes de conversion (cf. Tableau 5) ont été rédigées pour transformer les données du scénario vers le format compréhensible pour la librairie « jstree » et inversement.

| Modèle de données FORMID | Modèle de données jstree (sous-ensemble) |
|---|---|
| <pre>{<br>"nomExo":"Test",<br>"nomScenario":"ModeleVide",<br>"etapes":[<br>{<br>"idEtape":1,<br>"nomEtape":"\"\"",<br>"valeurEtape":1,<br>"numsControles":[1],<br>"etatInitial":"\"\"",<br>"consigne":"Consigne etape",<br>"messageSucces":"\"Succes etape\"",</pre> | <pre>{<br>"text":"Scénario = \"ModeleVide\"",<br>"state":{<br>"opened":true<br>},<br>"a_attr":{<br>"title":"ModeleVide",<br>"data-toggle":"tooltip"<br>},<br>"type":"scenar",<br>"data":"ModeleVide",<br>"children":[</pre> |

<table>
<tr>
<td>

```
        "messageEchec":"\"Echec
        etape\"",
        "etatFinal":"\"\"",
        "evaluation":"return true;",
        "typeTaches":null
     }
  ],
  "controles":[
     {
        "id":"c1",
        "nom":"TitreCtrl",
        "messageTrue":"\"Detection
        Controle\"",
        "valeurTrue":-1,
        "messageFalse":"\"Fin
        Detection\"",
        "valeurFalse":1,
        "evaluation":"return
        false;",
        "terminal":false,
        "praxeosPersos":null
     }
  ],
  "consigne":"Consigne globale",
  "messageSucces":"Succes
  exercice",
  "messageEchec":"Echec exercice",
  "variables":[],
  "simulName":"TPELEC",
  "simulConfig":null,
  "idAuthor":null,
  "groupLevel":"Seconde",
  "subject":"Electricité",
  "country":"France",
  "year":"2014"
}
```

</td>
<td>

```
      {
         "text":"Titre = \"Test\"",
         "type":"infoScenar",
         "data":"Test",
         "a attr":{
         "title":"Test",
         "data-toggle":"tooltip",
         "data-placement":"top"
      },
   […]
   {
   "text":"Étapes",
   "type":"listStep",
   "id":"e1",
   "children":[
      {
      "text":"Étape 1",
      "type":"step",
      "realOrder":0,
      "state":{
         "opened":false
      },
      "children":[
         {
         "text":"Nom = \"\"\"\"",
         "type":"infoStep",
         "data":"\"\"",
         "a attr":{
         "title":"\"\"",
         "data-toggle":"tooltip",
         "data-placement":"top"
         }
      ]
   } […]
}
```

</td>
</tr>
</table>

*Tableau 5 : Comparaison entre les modèles de données FORMID et jstree*

### 2. Compilation à la volée

Pour compiler un scénario, on se sert du fichier XML de description. Les données importantes en sont extraites grâce à un fichier XSL, celui-ci créant un fichier Java compilable. Les librairies de lecture et écriture de fichier XML ont été mises à jour, de même que le compilateur Java. Le fichier XSL a dû être modifié pour générer les noms adéquats pour les packages. Le résultat de la compilation est renvoyé au client, avec les messages d'erreur éventuels.

Les fichiers compilés sont placés directement sur le serveur dans un dossier portant le nom du scénario. Cependant, il ne peut être joué dans une séance de cours tant que l'exercice associé n'a pas été créé en base de données. Pour effectuer cette opération, une fonction « Publier » a été mise en œuvre.

### 3. Liberté vs. Sécurité

Un des objectifs d'un logiciel auteur est de favoriser la réutilisation de données. Il faut donc pouvoir ouvrir les scénarios existants, les modifier, réutiliser un exercice dans plusieurs séances de cours, utiliser la même configuration de départ pour une simulation dans

plusieurs exercices. Cependant, on ne peut garantir que tous les utilisateurs seront avertis et consciencieux, il faut mettre en place des garde-fous.

Le premier mécanisme consiste à ne pouvoir utiliser qu'un scénario publié. Lors de sa publication, un scénario est lié à un nouvel exercice, instancié en base de données. L'utilisateur ne peut ouvrir que des exercices existants. De plus, l'auteur a le droit de définir son scénario « privé ». S'il choisit cette option, l'exercice ne pourra pas être ouvert et édité par un autre auteur.

Le second mécanisme est plus systématique. Chaque création de fichier sur le serveur entraine le renommage (en y ajoutant un suffixe horodaté) d'un fichier du même nom, s'il existe. Cela vaut pour les fichiers XML de scénario, les dossiers contenant le code Java compilé et pour les fichiers d'initialisation de simulation. De cette manière, en cas de mauvaise manipulation d'un auteur, l'administrateur pourra toujours récupérer les fichiers écrasés.

### 4. Problème de l'initialisation d'un micromonde

Pour initialiser une simulation, un mécanisme d'état initial a été réfléchi. Un état de tous les éléments de la simulation est enregistré, il suffit de l'envoyer à la simulation en début d'étape pour la placer dans un état stable, celui désiré par l'auteur.

Dans un micromonde, l'utilisateur a le droit de créer et de supprimer des éléments à sa guise. Lorsqu'un état initial est poussé au micromonde, seuls les objets présents dans l'état initial enregistré par l'auteur sont mis à jour. Si des objets sont manquants, l'initialisation ne les créera pas, s'il y a des objets en plus, ils ne seront pas supprimés.

Un fichier de configuration initial a été prévu, il permet au démarrage de l'exercice de placer le micromonde comme l'a prévu l'auteur du scénario. Cela solutionne uniquement une partie du problème. Après chaque étape, l'auteur imagine l'état du micromonde uniquement avec les objets utilisés dans l'étape précédente. Si l'apprenant n'a pas respecté l'entièreté des consignes, il peut avoir mis un micromonde dans un état totalement différent. Il commencera donc l'étape avec une vue biaisée de l'exercice.

À terme, un fichier de configuration initiale pourrait être prévu par étape de l'exercice. Cela demanderait plus de travail pour l'auteur, mais garantirait la qualité de son scénario. De plus, cela permettra d'activer les actions de remise à zéro de l'étape et de passage à l'étape suivante, qui sont actuellement impossibles pour un micromonde. Cependant, on risque de perdre la continuité entre les étapes, cette suggestion est donc à analyser pour déterminer la meilleure solution.

# IV. FORMID – Évolutifs

Deux axes d'évolutifs étaient désirés dans FORMID. Le premier axe concerne le traçage des actions du tuteur. Il s'agit d'une part d'homogénéiser et de fiabiliser l'existant et, d'autre part, d'améliorer le modèle pour pouvoir l'exploiter plus facilement. Le deuxième axe vise à étendre l'interface de l'auteur pour le guider dans la conception et enrichir le scénario avec des informations liées au savoir et savoir-faire. Un troisième axe a été décidé au cours du stage et concerne l'internationalisation de l'interface.

## A. Traces des actions tuteur

Les traces récoltées lors d'un suivi de groupe constituent des données essentielles pour la recherche, en particulier pour évaluer l'adéquation des outils avec les besoins des enseignants. Elles s'avèrent extrêmement complexes dans leur définition, modélisation et mise en œuvre.

### 1. Réflexions sur le besoin

Les traces du module tuteur enregistrent toutes les actions qu'effectue un enseignant lorsqu'il suit un groupe d'apprenant. Pour comprendre sa manière de procéder, les chercheurs ont besoin que chaque action d'un tuteur soit contextualisée. Il faut répondre à la question : qu'y avait-il dans l'écran qui a pu inciter à faire cette action ? Outre les paramètres globaux, tel que l'élève sélectionné ou le moment de l'action, on s'intéresse à l'ordre d'affichage des élèves, au nombre de contrôles visibles à l'écran, au nombre d'élèves sélectionnés, ou encore au nombre d'erreurs de l'élève pour l'ensemble de l'étape. Nous avons identifié 45 actions et 38 paramètres, certains d'entre eux étant décomposés en liste de sous-paramètres.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Level | action | | details | | | | | |
| 2 | | open_session | | id_session | | | | | |
| 3 | | manual_request | | | | | | | |
| 4 | | auto_request_start | | time gap | | | | | |
| 5 | | auto_request_stop | | | | | | | |
| 6 | | adjust_time_gap | | new_time_gap | | | | | |
| 7 | | quit_session | | id_session | | | | | |
| 8 | Niveau 1 | Survol | Hover_Validation | id_learner, id_exercise,id_step, rang_validation_request, color,targeted_learner_displayed_global_info, targ | | | | | |
| 9 | | | Hover_Control | id_learner, id_exercise,id_step,id_control,rang_control, color, infos_targeted_learner_step, simul_info | | | | | |
| 10 | | | | | | | | | |
| 11 | Niveau 1 | SelectionTous | Select_all_learners | | | | | | |
| 12 | Niveau 1 | Deselection Tous | Unselect_all_learners | nb_selected_learners, list_selected_learners, list_selected_Learners_global_Info | | | | | |

*Figure 24 : Extrait de l'identification des actions et paramètres*

Il a fallu faire des choix sur la pertinence de chacun des paramètres en fonction de son impact sur les performances. Les traces devant transiter par le réseau, elles ne peuvent pas être trop volumineuses. De plus, une attention particulière a été portée au temps de création de la trace, pour qu'il n'affecte pas le temps de réaction de l'interface, pouvant entrainer un écran figé, ce qui déstabiliserait l'utilisateur. Par exemple, il est intéressant de connaitre le nombre d'échecs de l'élève affichés à l'écran tous contrôles confondus, mais il n'est pas nécessaire pour autant de tracer le détail de chaque échec.

Afin de garantir la qualité de la trace, des paramètres de fiabilité ont été proposés. Deux indicateurs ont été retenus : la précision temporelle et la cohérence [BERTI-EQUILLE12]. Pour le premier, trois marqueurs d'horodatage ont été placés, un à la création de la trace dans l'écran de l'utilisateur, un à l'arrivée sur le serveur et le dernier à l'enregistrement en

base de données. De cette manière, on peut surveiller la qualité du réseau et mesurer le rendement du serveur. Pour assurer la cohérence, des numéros de séquence sont générés dans l'interface et en base de données. On peut ainsi garantir la fiabilité des traces en comparant l'ordre des séquences avec la chronologie des marqueurs temporels.

Pour modéliser le résultat, un fichier de spécifications détaillé a été conçu. Il a permis d'en déduire le modèle des traces en base de données et dans l'application (cf. annexe I.). Il pourra servir par la suite de piste de réflexion sur ce qui est tracé et ce qui pourrait servir à d'autres recherches, il peut également servir de contrat d'interface avec d'autres logiciels.

| | global | time gap | id learner | info selected | global prog | info step | prev step | last step | ctrlId | rang_Val | rang_Ctrl | color | simul_info | seq_val | seq_ctrl | learner | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | targeted | | list selected | |
| | | | | | | | | | | | | | | | | Global | Detail | Global | Detail |
| open_session | X | | | | | | | | | | | | | | | | | | |
| manual_request | X | | | | | | | | | | | | | | | | | | |
| auto_request_start | X | X | | | | | | | | | | | | | | | | | |
| auto_request_stop | X | | | | | | | | | | | | | | | | | | |
| adjust_time_gap | X | X | | | | | | | | | | | | | | | | | |
| quit_session | X | | | | | | | | | | | | | | | | | | |
| N1_Hover_Validation | X | | X | | | X | | | | X | | X | X | | | X | | | |
| N1_Hover_Control | X | | X | | | X | | | X | | X | X | X | | | | | | |
| N1_Select_all_learners | X | | | | | | | | | | | | | | | | | | |
| N1_Unselect_all_learners | X | | | X | | | | | | | | | | | | | | | X |
| N1_Unselect_one_learner | X | | X | X | | | | | | | | | | | | X | | | X |
| N1_Add_learner_to_selection | X | | X | X | | | | | | | | | | | | X | | | X |
| N1_Select_one_learner | X | | X | X | | | | | | | | | | | | X | | | |
| N1_View_all_exercises | X | | | | | | | | | | | | | | | | | | |

*Figure 25 : Aperçu du fichier de spécification (cf. annexe I.)*

## 2. Plateforme Undertracks

Undertracks est une plateforme développée par l'équipe MeTAH, elle a pour objectif de capitaliser et traiter des données produites par des élèves ou de tuteurs qui utilisent des environnements EIAH. Le projet Undertracks se construit à partir des réalités du terrain, selon une démarche qualifiée de « bottom-up ». Cette stratégie de construction permet d'identifier des spécificités liées au monde des EIAH. L'objectif est de se laisser guider par les besoins des chercheurs en EAIH sans être contraint par les méthodes et outils classiques d'analyses de données [MANDRAN14].

Undertracks se distingue des autres plateformes sur trois points :

- Undertracks s'appuie sur un cycle de vie des traces qui commence par la spécification des hypothèses de travail, se poursuit par le design expérimental, la production et l'analyse des données et se termine par la valorisation des résultats et par l'émergence de nouvelles hypothèses.
- Undertracks permet de construire des opérateurs pour l'analyse des données en EAIH. L'originalité ici est de s'appuyer sur les besoins des chercheurs en didactique et d'un ingénieur en analyse de données pour co-construire des opérateurs des processus de traitements propres au secteur des EIAH. La structure informatique choisie permet de développer facilement des opérateurs élémentaires et de les combiner.
- Undertracks permet d'appliquer les différents opérateurs et processus créés sur des données parvenant de différents EIAH et ainsi de valider ces processus de traitements.

Les traitements qui peuvent être mis en place avec Undertracks sont nombreux : analyses exploratoires, validation d'hypothèses de recherche, filtrage, calculs (ex : fréquence d'une action, répartition des actions par utilisateur,…), recherches de récurrences, benchmarks, etc.

Il est intéressant d'insérer les traces produites par FORMID dans la base de données Undertracks. Cela permet d'identifier des patterns de comportement, par exemple les stratégies mises en œuvre par les tuteurs lors du suivi de la progression des élèves en recherchant des séquences d'actions dans les traces récoltées. Dans la figure 26, on observe le pattern « Sélection d'élève » - « Survol » (présent 95 fois dans les traces traitées), qui peut expliquer la stratégie d'un tuteur.

| Pattern | Frequency | Name |
|---|---|---|
| Survol->Survol | 1568 | SU_SU |
| Survol->SelectionTous | 30 | SU_ST |
| SelectionTous->Survol | 35 | ST_SU |
| SelectionEleve->SelectionEleve | 24 | SE_SE |
| SelectionEleve->Survol | 95 | SE_SU |
| Survol->SelectionEleve | 20 | SU_SE |
| SelectionEleve->SelectionTous | 14 | SE_ST |
| SelectionTous->SelectionTous | 58 | ST_ST |

*Figure 26 : Exemple de déduction de modèles de succession d'actions avec la fréquence associée*

Les données sur Undertracks sont structurées dans une base de données composée de cinq tables (cf. Figure 27). Pour qu'une trace soit compatible avec Undertracks, elle doit fournir les informations de deux tables minimum : « Event » et « Description ». Les traces FORMID sont donc transformées puis exportées dans un fichier Excel, pour correspondre au format d'import d'Undertracks. Cette fonction est accessible aux administrateurs FORMID.

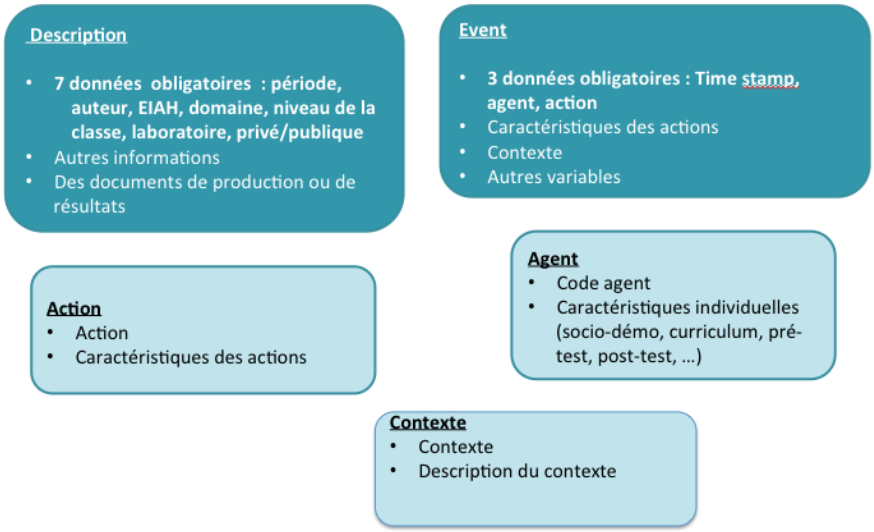



*Figure 27 : Tables de la base de données Undertracks*

Undertracks possède aussi une interface JavaScript pour insérer des traces à la volée dans la base de données. Dans notre cas, cette façon de faire n'a pas été retenue, notamment pour des problèmes de performance. Comme les traces doivent être insérées dans la base

FORMID, faire deux appels de sauvegarde, un sur chaque logiciel, risquait de saturer le réseau.

### 3. Adaptation de la base de données

L'insertion du nouveau modèle de traces a demandé la création de 9 tables en base de données (cf. annexe B.). L'objectif était de normaliser le schéma, pour éviter de devoir placer des listes sous forme de chaine de caractères dans une colonne. Pour permettre de garder le même modèle pour des actions qui prennent un nombre distinct de paramètres de contexte, les colonnes peut être vides. De cette manière, une trace de connexion qui prend très peu de paramètres pourra se trouver dans la même table qu'une trace de survol qui a de nombreux paramètres de contexte associés.

Le paramètre de contexte le plus fin est la progression d'un élève sur une étape ou sur un contrôle. La progression est définie par le nombre de succès, le nombre d'échecs, le nombre d'événements neutres et le total de ces trois catégories. Cette progression peut être rattachée à différents paramètres. Si le paramètre qui nous intéresse est lié à l'élève et à l'étape, il contiendra la progression des demandes de validation pour cet élève et cette étape ainsi que la progression par contrôle lié à l'étape. Si le paramètre est lié à au groupe, la progression contiendra le total de toutes les demandes de validation des élèves visibles à l'écran, ainsi que le total de tous les contrôles visibles à l'écran tout élèves confondus.

### 4. Traçage JavaScript

#### *a. Récolte des données*

L'enregistrement d'une trace se déroule via un seul appel au serveur. L'objet à tracer est au format JSON, il est plus moins complet en fonction du type d'action. Il est initié toujours de la même façon, avec les données contextuelles (utilisateur, nom de l'action, moment de l'action, séance suivie …). Un chronomètre est mis à jour, il permet de calculer le temps entre deux actions. La trace est initialisée au début de l'action, pour garder le moment précis de l'intention de l'utilisateur. Elle est envoyée en fin de méthode, afin de pouvoir calculer les champs mis à jour durant le traitement, tel que le niveau d'arrivée, la donnée mise à jour, le groupe d'élèves sélectionnés…

Les éléments de suivi dans l'interface sont les petits rectangles matérialisant l'observation d'une situation remarquable (contrôle), ainsi que les grands rectangles matérialisant les demandes de validation (cf. I-A-3). Ils constituent de fait des indicateurs de progression des élèves. Lors du survol de l'un d'entre eux, une fenêtre « pop-up » affiche les caractéristiques de l'état dans lequel l'élève correspondant avait placé la simulation au moment où l'indicateur a été inséré dans l'interface de suivi. Lors de l'apparition de cette fenêtre, un événement est lancé, pour mettre à jour un chronomètre. Un deuxième événement est lancé à la disparition de la fenêtre. Si le temps du chronomètre est inférieur à une demi-seconde, la trace du survol par le tuteur ne sera pas générée, car on considère que le tuteur n'a pas eu le temps de se servir de l'information. Si la fenêtre est restée affichée suffisamment longtemps, la trace générée contient le détail de la trace élève (informations affichées dans la fenêtre pop-up).

### b. Agrégations à la volée

Dans un souci de performance, on n'enregistre jamais dans la trace tuteur l'ensemble des traces élève matérialisées sur l'écran, mais le nombre total de traces, le nombre de traces en succès, en erreur ou neutre. Les traces élève matérialisées dans l'écran sont rangées dans des tableaux dans le modèle de données (construit en JavaScript et stocké dans la session du navigateur). Les traces de validation sont au niveau de l'étape liée à l'élève, les traces de situations à observer sont au niveau du contrôle lié à l'étape. Pour éviter de devoir parcourir toutes les traces lors du calcul de la progression, des totaux sont déjà présents aux différents niveaux, et sont mis à jour à l'arrivée d'une nouvelle trace élève.

```
//ajout de la trace dans le modèle
stepInfo.traces.push(trace);
var ctrlIdx = trace.controlek;

//mise à jour des agrégats
//nombre total de traces concernant ce contrôle
$scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id][ctrlIdx
].nb_all++;

//Si c'est une trace de succès
if (trace.resultat == 1) {
      //Mise à jour du nombre de succès pour ce contrôle
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].nb_green++;
      //Mise à jour de la succession de type de traces (ex. VVRV)
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].seq += "V";
//Si c'est une trace d'échec
} else if (trace.resultat == -1) {
      //Mise à jour du nombre d'échec pour ce contrôle
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].nb_red++;
      //Mise à jour de la succession de type de traces (ex. VVRV)
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].seq += "R";
//Si c'est une trace neutre
} else {
      //Mise à jour du nombre de neutre pour ce contrôle
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].nb_orange++;
      //Mise à jour de la succession de type de traces (ex. VVRV)
     $scope.aggregateInfos[studentInfo.id][exoInfo.id][stepInfo.id]
     [ctrlIdx].seq += "O";
}
```

*Extrait de code 15 : Mise à jour des totaux lors de l'arrivée d'une trace élève de contrôle*

Lors de la production d'une trace tuteur nécessitant des informations de progression de l'élève ou du groupe, on détermine quel est le niveau de précision recherché par rapport à l'action. Si c'est la progression globale de l'élève, on va parcourir tous les exercices et les étapes visibles de l'élève, et additionner les totaux de chacune des étapes et chacun des contrôles pour obtenir le total général pour cet élève. Si par contre on a besoin des informations pour une étape sans distinction d'élève, on itère sur les élèves visibles, et pour chacun, on récupérera la progression sur l'étape visée pour en faire un total général.

## B. Module auteur « assisté »

L'extension du module auteur est motivée par un double besoin.

Le premier vise à formaliser dans chaque scénario produit les choix didactiques dont il résulte. Ces choix didactiques (problème à résoudre, connaissances mises en jeu, erreurs fréquentes méritant d'être observées, progression d'un exercice à l'autre) n'étaient jusque-là exprimables qu'implicitement via la modélisation des situations à observer (contrôles) et des conditions de validations d'étapes. Le rapprochement constaté entre ces informations telles qu'exprimées sur papier par les enseignants lors de la conception d'un scénario et les concepts praxéologiques (cf. chap. I-D-2) a conduit en un premier temps à étendre le modèle de scénario avec ces concepts. L'objectif est ici de favoriser la réutilisation des scénarios par l'explicitation des choix qui en sont à l'origine.

Le second vise à assister l'auteur dans sa tâche de conception. L'interaction du module auteur avec un entrepôt de représentations praxéologiques liées à une discipline et un niveau d'études donnés peut offrir à l'auteur (et en particulier aux jeunes enseignants) des pistes et des recommandations qui le guident au cours de son travail de conception. En effet, dans la mesure où ces informations sont issues de référentiels fournis par l'institution ou de connaissances empiriques basées sur l'expérience didactique et pédagogique de ses pairs, l'enseignant auteur peut s'en servir comme guide tout au long de la conception. L'opportunité du projet OntoPrax (cf. chap. I-D-2), conduisait naturellement à connecter le module auteur avec des praxéologies représentées et organisées dans cette application.

### 1. Rapprochement entre le scénario pédagogique FORMID et les praxéologies

La modélisation praxéologique des savoirs et savoir-faire enseignés à un élève et attendus de lui par une institution (c'est-à-dire la modélisation des éléments concernant l'apprentissage), se divise en plusieurs entités (cf. Figure 28) réparties en deux blocs. Le premier est d'ordre pratique (savoir-faire) et est constitué de types de tâches (ce que doit faire l'élève), qu'on réalise (relation « est réalisé par » dans la Figure 28) par des techniques (autrement dit, des méthodes de résolution). Un type de tâches est au minimum constitué d'un verbe d'action et porte (relation « porte sur » dans la Figure 28) sur des entités d'une discipline. Chaque technique est composée (relation « est constituée de » dans la Figure 28) de types de tâches qui sont autant d'étapes nécessaires pour arriver à la résolution du type de tâches. Le deuxième bloc est d'ordre théorique et se compose d'une technologie (savoirs) qui justifie (relation « justifie » dans la Figure 28) une technique.

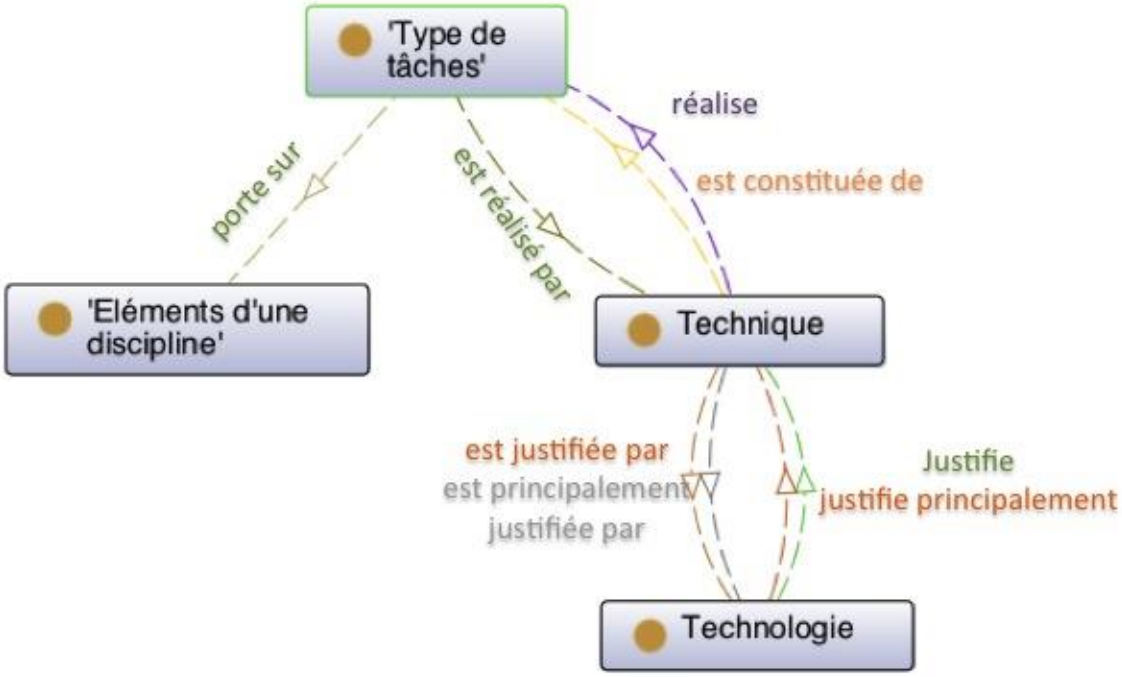



*Figure 28 : Modèle praxéologique de la connaissance*

Par exemple, un type de tâche pourrait être « Calculer la valeur de la résistance de protection dans un circuit en série contenant des lampes identiques, connaissant les caractéristiques des autres dipôles. ». Le verbe d'action est ici « Calculer » et les éléments de la discipline (ici le domaine de l'électricité en physique) sont : résistance de protection, circuit en série, lampes (dipôles). Une technique utilisée est l'application de l'équation « U = R.I » (U pour la tension, I pour l'intensité du courant et R pour la résistance) justifiée principalement par la technologie globale de « la loi d'Ohm ». La résolution de l'équation en utilisant cette technique entraine l'application d'autres types de tâche dont « le calcul de l'intensité qui traverse une résistance » et « le calcul de la tension aux bornes de la résistance ». Les deux types de tâches précédant admettent aussi des techniques consistant à appliquer des technologies représentées par les lois de Kirchhoff.

Dans un scénario FORMID, chaque étape peut être vue comme un type de tâche dans la mesure où elle correspond à un problème que l'élève doit résoudre en interagissant avec la simulation choisie pour cette étape, avec une méthode de résolution attendue, en mettant en œuvre des savoirs qu'il est supposé avoir. Le modèle ne juge pas de la justesse des techniques, une technique pouvant être considérée comme fausse par l'enseignant, ou non pertinente par rapport à l'énoncé, ou encore d'un niveau différent que celui attendu (utilisation du passé antérieur dans une classe de CP par exemple). Ces techniques erronées peuvent être repérées lors de leur mise en œuvre via la définition de contrôles dans le scénario FORMID, l'explicitation des contrôles pouvant à son tour être embarquée dans le scénario grâce à l'extension du modèle de scénario.

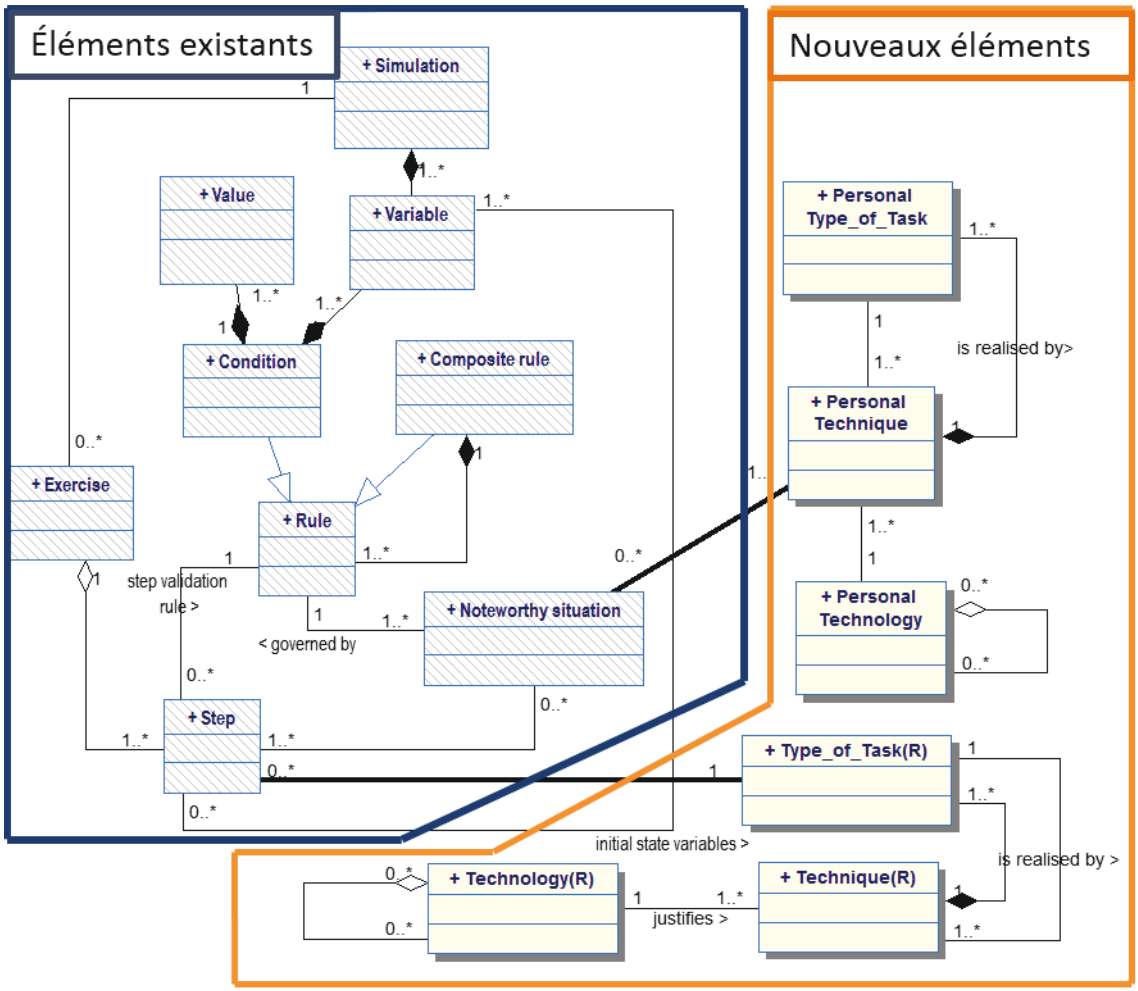



*Figure 29 : Modélisation simplifiée de l'enrichissement du scénario*

## 2. Représentation d'éléments en physique dans le logiciel OntoPrax

OntoPrax est un référentiel sous forme d'ontologie des compétences (savoir et savoir-faire) attendues par l'institution au niveau du socle commun de l'école primaire en Français et en Mathématiques [Cartographie des savoirs14]. Il est développé par l'équipe MeTAH avec des partenariats externes. Chaque élément praxéologique y trouve une place définie, et un calcul de hiérarchisation des types de tâche est réalisé, en partant de la tâche la plus générique vers la plus spécifique. Par exemple, le type de tâche « additionner deux entiers » est un cas particulier de « additionner deux décimaux » du fait qu'il partage la même nature d'opération (le fait d'additionner), que les entiers sont des cas particuliers de décimaux et que les cardinalités sont les mêmes (deux entiers pour deux décimaux). Les entités praxéologiques sont rattachées à un programme qui factorise les informations suivantes : un niveau scolaire, une discipline (dans l'exemple ci-dessus, il s'agit des Mathématiques), un pays et une date de publication (en effet, le programme des cours varie en fonction de l'institution).

OntoPrax étant un modèle générique, nous avons pu modéliser et implémenter en son sein des praxéologies en rapport avec des travaux pratiques autour de la loi d'Ohm (en Physique). L'objectif est d'étudier la faisabilité d'une liaison avec FORMID sur la base des scénarios existants.

### 3. Intégration au module auteur

Autour d'OntoPrax, une API d'interrogation est fournie. Elle expose une interface REST qui permet à des logiciels extérieurs d'accéder et d'interroger les données ontologiques. L'URL à appeler est construite suivant une syntaxe composée de mots-clés et de paramètres qui permettent de définir le besoin et les filtres à mettre en œuvre, la réponse est du type JSON.

L'URL se décompose de cette manière :

- http://serverName/knowledgemapping/webresources/
- Point d'entrée (« typedetaches », « techniques », …)
- Version (actuellement non traitée)
- « /select/ »
- Données recherchées ou filtre (« typedetaches », « techniques », ou « nomDeTechnique », « nomDeDiscipline »)
- Données recherchées si filtre à l'étape précédente (« madeOf », « badtechniques », …)
- Filtres supplémentaires (Pays, année, niveau, …)

Dans FORMID, l'utilisation des données de l'ontologie, et donc de l'aide à la conception, est facultative. De plus, nous ne possédons pas d'ontologie dans toutes les disciplines et tous les domaines. Pour déterminer si l'aide est disponible, l'auteur est invité à définir le contexte du scénario : pays, niveau scolaire, discipline. Sur base de ces informations, on peut déterminer si une ontologie est déclarée, et si c'est le cas, la proposer à l'auteur. Le chargement des données de l'ontologie se fait à la création du scénario, ou au chargement d'un scénario existant. Tous les types de tâche sont rapatriés via l'URL :

*« http://serverName/knowledgemapping/webresources/typedetaches/v1/select/Physics/France/2008 »*

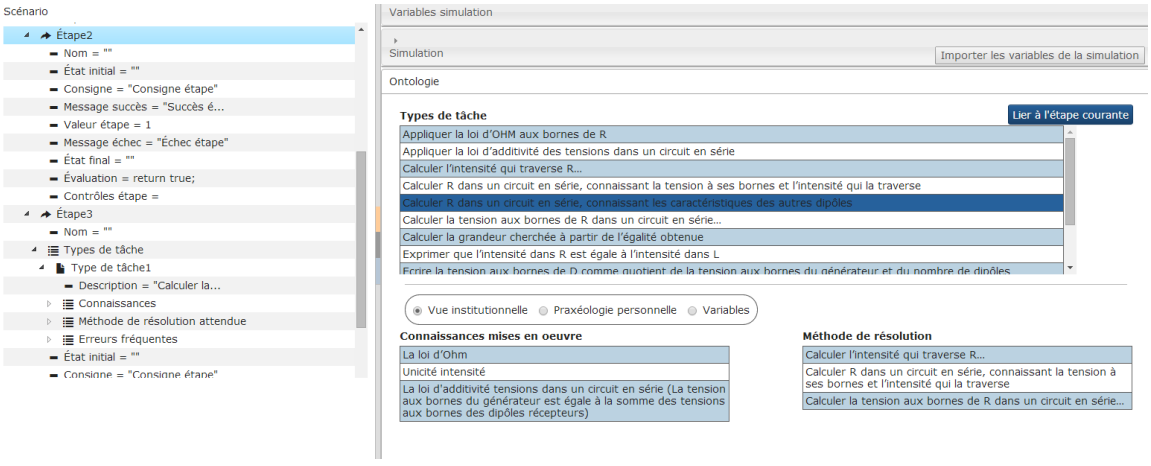



*Figure 30 : Écran d'aide à la conception grâce à des informations praxéologiques*

Pour chaque type de tâche, on va ensuite chercher sa technique et sa méthode de résolution, ses technologies et les erreurs fréquentes pour ce type de tâche. L'auteur peut naviguer dans les résultats pour trouver une correspondance avec les étapes de son scénario, s'inspirer des informations affichées pour améliorer sa conception, et finalement

lier les éléments de l'ontologie au scénario en les important dans l'étape pour un type de tâche, ou dans un contrôle pour une erreur fréquente.

Une fois importés, les informations seront sauvegardées dans le XML, et pourront être utilisées même si la liaison à l'ontologie n'est pas disponible. Les éléments donnant du sens au contexte de l'exercice, la discipline, le niveau scolaire, le pays d'enseignement, l'année du programme institutionnel sont sauvegardés à la racine de l'exercice même si l'auteur n'utilise pas l'aide à la conception.

### 4. Pérennité de la solution

Pour ne pas être dépendant d'un seul logiciel d'ontologie, certains mécanismes ont été mis en place. Une table d'ontologie a été créée en base de données, elle est liée à la hiérarchie Pays-Niveau-Domaine, ce qui permet d'avoir des serveurs d'ontologie différents pour chaque matière ou niveau scolaire. La table d'ontologie contient une URL d'appel et une version, pour permettre de se rattacher à un programme institutionnel particulier si besoin.

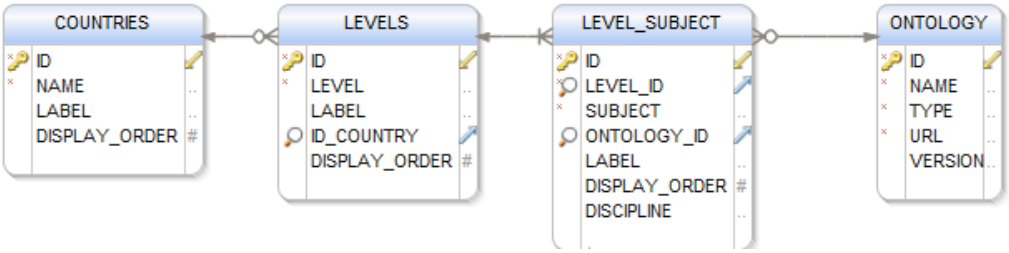



*Figure 31 : Hiérarchie de description d'une ontologie*

Du côté JavaScript, un principe d'héritage a été mis en place. La « classe » Ontologie possède quelques attributs de fonctionnement, telle que l'URL du serveur d'ontologie et la version, récupérée de la base de données, le niveau, récupéré des informations entrées par l'auteur, et une méthode à enrichir, « getData », qui attend une structure de données précise (cf. Extrait de code 16).

```
{
  "idDeLaTache1" : {
            "id" : "idDeLaTache1",
            "description" : "desccriptionDeLaTache1",
            "techniques" : {
                  "technique1" : {
                        "labelResolution1" : "descResolution1",
                        "labelResolution2" : ...
                              },
                  "technique2" : ...
                        },
            "technologies" : {
                  "technologie1" : "descriptionTechnologie1",
                  "technologie2" : ...
                         },
            "badTechnics" : {
                  "erreurFrequente1" : "descErreurFrequente1",
                  "erreurFrequente2" : ...
                        }
              },
  "idDeLaTache2" : ...
}
```

*Extrait de code 16 : Format attendu par FORMID pour les données ontologiques*

Il faut ensuite étendre cette classe pour transformer les réponses du serveur d'ontologie afin de correspondre aux attentes de FORMID. Dans le cas d'OntoPrax, la méthode « getData » nettoie une partie des retours, OntoPrax fournissant beaucoup de données non utiles pour FORMID. Des appels récursifs sont réalisés pour aller chercher les informations concernant la méthode de résolution d'un type de tâche, et les erreurs fréquentes.

Chaque serveur d'ontologie doit posséder sa classe et celle-ci doit être déclarée dans la base de données. Lorsque le JavaScript reçoit la configuration de l'ontologie au chargement

```
var currentOntology = eval("new " + data.ontology.type + "('" +
data.ontology.type + "','" + data.ontology.url + "','" +
data.discipline.name + "','" + data.studentLevel.name + "','"
+ data.ontology.version + "')" );
var ontologyData = currentOntology.getData() ;
```

*Extrait de code 17 : Instanciation dynamique d'une classe de récupération des données pour un serveur d'ontologie ciblé.*

de l'écran, il instancie la nouvelle classe de traitement ontologique de manière dynamique et appelle la méthode « getData » pour alimenter l'écran d'aide à la conception.

### C. Internationalisation

Un constat rapidement réalisé durant les démonstrations du projet, était que le vocabulaire utilisé par les chercheurs n'était pas le même que celui des ingénieurs, et que les termes utilisés dans les écrans étaient souvent incorrects ou pas assez explicites. Une solution d'internationalisation a été mise en œuvre pour permettre aux chercheurs de modifier facilement tous les labels. Un autre avantage de cette solution est la possibilité de passer

d'une langue à une autre, les publications se faisant le plus souvent en anglais et les expérimentations ont jusque-là eu lieu dans des classes françaises.

Les fichiers de langue se trouvent dans le dossier de configuration du serveur, ils sont composés d'une liste de clé/valeur. La norme choisie est « nomEcran.label[.fonction] ». La langue est chargée au premier appel à FORMID, ou suite à modification de la langue via le menu. Ces appels pouvant se faire hors connexion, la méthode est publique.

Les dictionnaires sont gardés en session, ils sont utilisés par une directive AngularJS. L'attribut « i18n : 'clé' » sur une balise HTML remplace l'intérieur de la balise par la valeur liée à la clé. Si on veut utiliser la valeur dans un autre contexte, on peut utiliser l'attribut « i18n-attr » et la barre verticale. Par exemple, pour le texte de survol d'une image, la balise sera « <img src='xxx' i18n-attr='cléTxt|title'/> ».

# Conclusions

En conclusion, nous revenons sur les différents aspects du stage en établissant une synthèse des objectifs et du travail effectué puis en proposant quelques perspectives d'évolutions. Enfin un bilan personnel termine ce mémoire.

## A. Conclusion

### Apport des choix technologiques et de la nouvelle architecture

La suite logicielle FORMID nécessitait des corrections liées notamment à la technologie utilisée. Le choix de transformation d'architecture était ambitieux par la complexité de migration et les risques inhérents aux changements, mais a permis de se libérer des contraintes de la JVM. La remise au propre du code, la formalisation des traces système et le choix de technologies faciles d'accès pour un développeur et demandant le minimum de programmation, donnent aujourd'hui à FORMID un socle solide pour ses futures évolutions. La centralisation des ressources sur un serveur unique facilite la maintenance et la détection d'erreurs. La récence et la robustesse des librairies laissent présager un bel avenir sans problèmes majeurs de compatibilité, tout en améliorant les performances et la sécurité. Cela permet d'autre part d'offrir des opportunités de stage non plus uniquement attrayantes par le sujet, mais aussi par les technologies utilisées. En effet, la plupart des étudiants en informatique cherchent avant tout à acquérir une expérience dans les technologies de pointe qui sont porteuses sur le marché de l'emploi.

Auparavant, pour faire tourner la suite FORMID :

- Il fallait
  - La plateforme « testbed », équipée du logiciel « LDI », pour exécuter FORMID élève et FORMID tuteur, en tant qu'applets Java.
  - Un serveur Apache pour faire tourner uniquement en local le module FORMID Auteur.
- Il n'y avait pas d'interface d'administration (constitution des classes, affectation des rôles, nom des élèves, composition des cours, …)
- Les traces élève étaient enregistrées dans une base de données « eXists » qui ne pouvait être accédée qu'à partir du port 8080 et les traces tuteur n'étaient pas en base de données.

Aujourd'hui :

- La suite tourne entièrement sur un serveur applicatif (JBoss) et est accompagnée d'un module d'administration.
- Le serveur est déployé sur une machine virtuelle et pourrait être déplacé sur un poste local, sur une autre machine virtuelle ou sur un serveur physique.
- Les traces de chacun des modules sont enregistrées dans une base de données unique, intégrée au serveur applicatif.

De fait, tous les problèmes de sécurité et de compatibilité liés aux anciennes technologies et architecture sont résolus. L'administration du logiciel est facilitée par des interfaces web. L'ensemble des trois modules de la suite FORMID peut être exécuté dans des

configurations diverses, sans problème d'accès aux ressources locales du poste ou de politique de sécurité informatique des établissements.

### Enrichissement de la suite FORMID

Du point de vue fonctionnel, aucune perte n'est à déplorer. La suite FORMID s'est au contraire enrichie d'une part d'un traçage de qualité, et d'autre part d'une extension significative du module auteur.

La spécification et l'implémentation de traces de qualité pour le module de suivi apporte une valeur ajoutée non seulement au projet, mais à l'ensemble de l'équipe MeTAH. Pour FORMID, l'analyse de ces traces va pouvoir alimenter des hypothèses de recherche sur la stratégie suivie par les tuteurs dans leur activité de suivi. Les marqueurs de qualité définis dans le modèle permettront de s'assurer de la fiabilité de la production des données. L'équipe quant à elle pourra se baser sur ce travail pour établir dans les autres projets de nouvelles façons de concevoir les traces, pour s'assurer de l'homogénéité et de la qualité de celles-ci.

D'autre part, l'apport de ressources praxéologiques dans l'écran de conception permet désormais à l'auteur de mieux cibler les scénarios qu'il crée, de s'assurer que ce qu'il propose est bien en phase avec les préconisations de l'institution, et de s'appuyer sur une base de connaissances communes sur le type d'erreurs fréquemment commises par les élèves sur le type d'exercices qu'il modélise. Chaque scénario peut donc être enrichi avec ces informations, ce qui en favorisera la réutilisation, en offrant la possibilité de mieux comprendre le type de tâche que son auteur a voulu mettre en œuvre et les connaissances qui y sont associées.

### Industrialisation

L'axe d'industrialisation apporté durant le stage offre un vrai gain à l'équipe, avec une centralisation du code et un suivi des modifications, ce qui évite les pertes de données et les incohérences des versions. Le rassemblement de la documentation sur le gestionnaire de source permet à un nouveau collaborateur dans l'équipe de prendre en main le projet sans être dépendant des disponibilités des chercheurs. Enfin, la mise en place d'une machine virtuelle offre la possibilité d'avoir les fonctionnalités d'un serveur sans en subir le coût, avec des outils de sauvegarde automatique et de sécurité gérés par les administrateurs système.

La rédaction d'un dossier d'architecture technique complet, accompagnée de sources documentées et de scripts d'automatisation, permet aux chercheurs et aux futurs intervenants de reprendre facilement le travail pour d'éventuels correctifs ou pour des évolutions. Les choix de librairies avec une communauté active et nombreuse offrent la sécurité d'une documentation en ligne variée et adaptée à tous les cas de figure.

## B. Perspectives et évolutions

La suite logicielle FORMID a encore de nombreuses possibilités non exploitées, et offre de nombreuses perspectives pour de futurs stages.

### 1. Variétés des simulations

La nouvelle architecture n'a été testée qu'avec une seule simulation (en réalité un micromonde), TPElec. Il serait intéressant de brancher FORMID avec d'autres simulations, dans des domaines variés. La compatibilité est normalement assurée. Un projet de l'équipe MeTAH porte sur des objets tangibles en mathématiques, c'est-à-dire que les élèves manipuleraient des objets physiques connectés via la technologie « Bluetooth » à une tablette. FORMID pourrait se connecter à ce dispositif pour créer des scénarios, enregistrer le travail des élèves et permettre à l'enseignant de suivre leur avancement.

### 2. Enrichissement de la communication avec les ontologies

L'ontologie utilisée pour concevoir les écrans du module auteur a été alimentée avec peu d'éléments, l'objectif étant de montrer la faisabilité. Plusieurs axes d'amélioration peuvent maintenant être envisagés, parmi lesquels :

- Alimentation de l'écran avec des variables liées au type de tâche. Le professeur peut dès lors faire évoluer son scénario en fonction de ces variables, en sachant qu'elles pourraient soit influencer la méthode de résolution classique, soit amener l'élève à commettre des erreurs.
- Amélioration du système de filtres. Pour une ontologie complexe, le choix du pays, du niveau et de la discipline ne sera pas suffisant. Il faudrait permettre à l'auteur de préciser sa recherche par domaine ou par thème.
- Utilisation des données praxéologiques enregistrées dans le scénario pour assister le tuteur dans ses tâches de suivi. Il verrait ainsi lors du survol d'une étape ou d'un contrôle des informations sémantiques qui l'aideraient à mieux comprendre le scénario et à mieux inférer le raisonnement de l'élève.

### 3. Utilisation des traces des trois modules

La consolidation des traces tuteur permet maintenant de pouvoir les exploiter sereinement. La compatibilité avec Undertracks va permettre d'enrichir la base Undertracks et d'analyser plus finement les stratégies d'utilisation de FORMID par les tuteurs. Il serait intéressant de croiser ces analyses avec celles des traces élève et des futures traces auteur, pour voir si des patterns transversaux peuvent être déduits, par exemple des dépendances entre une stratégie d'utilisation élève qui provoquerait des actions récurrentes chez le tuteur, ou encore si le nombre d'étapes dans un exercice a une influence sur le nombre d'erreurs de l'élève ou la stratégie de suivi du tuteur.

## C. Bilan personnel

Mon objectif premier durant ce stage était d'augmenter mon expertise technique en proposant, comparant et évaluant différents choix techniques. J'ai pu faire plus que cela, car mon rôle dans le stage a couvert l'entièreté d'un cycle classique de projet, avec un accent prononcé sur les aspects techniques. J'ai participé au recueil du besoin et à sa formalisation, j'ai effectué des études de faisabilité et l'implémentation des choix techniques. J'ai posé une architecture technique complète, évolutive, documentée, j'ai migré le code et les données existantes et j'ai développé de nouveaux composants. J'ai

réalisé des tests unitaires et des tests fonctionnels. Enfin, j'ai rédigé un dossier d'architecture technique détaillé, permettant la reprise complète du projet par une autre personne. Ce stage m'a ainsi permis aussi de monter en compétence sur les dernières technologies, sur leurs avantages, sur leurs inconvénients et aussi sur les écueils à éviter. Cela me sera sans nul doute profitable pour mes prochaines expériences.

Effectuer mon stage au sein d'une équipe de recherche m'a beaucoup apporté en termes de méthodologie et de prise de recul. La confiance que Mesdames Guéraud et Lejeune m'ont accordée m'a stimulé par la grande liberté qu'elle me donnait dans l'organisation de mon travail, tout en m'obligeant à être plus rigoureux, persévérant et autonome. Alors que les contraintes du secteur privé et, en particulier, le rythme de développement selon des méthodes agiles demande une grande réactivité mais peu de remises en question, j'ai pu apprendre à prendre le temps nécessaire au test de plusieurs solutions afin de sélectionner la plus adaptée au projet et, pour résumer, à « réfléchir avant d'agir ». De plus, le fait de travailler au plus près des commanditaires de mon projet, sans autre intermédiaire quant à la gestion de ce dernier était pour moi une expérience nouvelle que j'ai vraiment appréciée. Enfin, le contexte pluridisciplinaire de l'équipe MeTAH et le cœur de ses recherches centrées autour de l'apprentissage humain, m'ont permis de participer à des échanges très enrichissants sur le plan personnel.

Dans ce contexte, je pense avoir amené à l'équipe et au projet un point de vue différent, plus proche du métier informatique, en centrant l'attention sur des sujets comme les performances ou la cohérence des données, sur la pérennité de la solution technique et les processus d'industrialisation. Sur ce dernier point, j'aurais aimé aller plus loin en mettant en place des processus de tests unitaires automatisés. Cependant, la charge de travail résultant de la migration technologique et les priorités du projet ne m'en ont pas laissé le temps.

Au final, ce stage aura donc été très enrichissant, et m'aidera certainement dans ma future carrière.

## Glossaire

| | |
|---|---|
| Applet Java | Petit programme, écrit dans le langage informatique Java, et inséré dans une page web. L'objectif est de fournir des fonctionnalités que ne peut offrir le HTML [Wikipedix13]. |
| Flash | Technologie devenue une des méthodes les plus populaires pour ajouter des animations et des objets interactifs à une page web ; de nombreux logiciels de création et OS sont capables de créer ou d'afficher du Flash. Flash est généralement utilisé pour créer des animations, des publicités ou des jeux vidéo. Il permet aussi d'intégrer de la vidéo en streaming dans une page. |
| Framework | Ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture) [HASHARBOT03]. |
| Java | Java est un langage de programmation orienté objet, Open Source, détenu et maintenu par la société Oracle. |
| Micromonde | Représentations informatiques simplifiées de la réalité, dans lesquelles l'utilisateur possède une grande autonomie et où chacune de ses actions provoque l'effet qu'il aurait dans le monde réel. Cela permet de mettre en situation l'élève pour réaliser des travaux pratiques sans danger. |
| Ontologie | Spécification formelle d'une conceptualisation partagée [PSYCHE03]. C'est une représentation en langage informatique d'une vision d'un concept commune à un groupe. |
| Simulation | Modélisation informatique d'un phénomène ou d'une expérience scientifique. Cela permet de faire varier les caractéristiques des objets impactés et d'en observer les conséquences. |
| Praxéologie | Discipline qui étudie l'action humaine, et tente de la modéliser. |
| XML | Langage de structuration de données à l'aide de balises. |
| XSL | Langage de transformation qui permet de parcourir un document XML et de créer un fichier en sortie, XML ou non. |

# Bibliographie


[Adobe] Adobe. ActionScript® 3.0 Reference for the Adobe® Flash® Platform [en ligne]. Disponible sur : http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/external/ExternalInterface.html (consulté en Avril 2014).

[AngularJS14] Google. AngularJS [en ligne]. Disponible sur : https://angularjs.org/ (consulté en Avril 2014).

[Atmosphere Framework14] Async-io. Atmosphere Framework [en ligne]. Disponible sur : http://async-io.org/ (consulté en Avril 2014).

[BEAUD14] BEAUD Valéry. Client-serveur [en ligne]. , 2014 Disponible sur : http://fr.wikipedia.org/wiki/Client-serveur (consulté en Avril 2014).

[BERTI-EQUILLE12] BERTI-EQUILLE Laure. La qualité et la gouvernance des données au service de la performance des entreprises. France : Hermes Science , 2012, 25-36 p.
ISBN 978-2-7462-2510-7

[Bootstrap14] Twitter. Bootstrap [en ligne]. Disponible sur : http://getbootstrap.com/ (consulté en Avril 2014).

[BORDERIE03] BORDERIE Xavier. L'architecture REST [en ligne]. , 2003 Disponible sur : http://www.journaldunet.com/developpeur/tutoriel/xml/030707xml_rest1a.shtml (consulté en Avril 2014).

[BORST97] BORST Willem Nico , "Construction of Engineering Ontologies for Knowledge Sharing and Reuse," Université de Twente, Enschede, Thèse 1997.

[CAGNAT07] CAGNAT Jean-Michel. (2007) FORMID_Doc.

[Cartographie des savoirs14] LIG. Cartographie des savoirs [en ligne]. Disponible sur : http://www.cartodessavoirs.fr/index.php?option=com_content&view=article&id=50&Itemid=470&lang=fr (consulté en Avril 2014).

[CAVAZZA14] CAVAZZA Frédéric. 20 ans d'évolution des IHM web [en ligne]. , Janvier 2014 Disponible sur : http://www.interfacesriches.fr/ (consulté en Avril 2014).

[CentOS14] Red Hat. CentOS [en ligne]. Disponible sur : http://www.centos.org/ (consulté en Avril 2014).

[CHAACHOUA10] CHAACHOUA Hamid , "La praxéologie comme modèle didactique pour la problématique EIAH. Etude de cas : la modélisation de la connaissance des

élèves.," Université Joseph Fourier, Grenoble, Mémoire d'habilitation à Diriger des Recherches 2010.

[CHEVALLARD06] CHEVALLARD Yves. La théorie anthropologique des faits didactiques devant l'enseignement de l'altérité culturelle et linguistique : Le point de vue d'un outsider [en ligne]. , Mars 2006 Disponible sur : http://yves.chevallard.free.fr/spip/spip/IMG/pdf/La_TAD_devant_l_alterite_culturelle_et_linguistique.pdf (consulté en Avril 2014).

[COHEN07] COHEN Emmanuel. Mov'AMP [en ligne]. , Août 2007 Disponible sur : http://fr.wikipedia.org/wiki/Mov'AMP (consulté en Avril 2014).

[COPE13] COPE Greg. Future of Java Applets [en ligne]. , 2013 Disponible sur : http://www.algosome.com/articles/future-of-java-applets.html (consulté en Avril 2014).

[DERKSEN14] DERKSEN Bryan. Unix time [en ligne]. , 2014 Disponible sur : http://en.wikipedia.org/wiki/Unix_time (consulté en Avril 2014).

[ECHTERBILLE13] ECHTERBILLE Patrick , "Panorama de système auteur et d'assistance à la conception de solutions pédagogiques : modèles, architectures, environnements, technologies, types d'utilisateurs sous-jacents," Grenoble, 2013.

[FileZilla] FileZilla Project. FileZilla [en ligne]. Disponible sur : https://filezilla-project.org/ (consulté en Avril 17).

[Forge IMAG14] IMAG. Forge IMAG [en ligne]. Disponible sur : https://forge.imag.fr/ (consulté en Avril 2014).

[FusionForge14] FusionForge. FusionForge [en ligne]. Disponible sur : http://fusionforge.org/ (consulté en Avril 2014).

[Git -- everything-is-local14] Git. Git --everything-is-local [en ligne]. Disponible sur : http://git-scm.com/ (consulté en Avril 2014).

[GONTOVNIKAS13] GONTOVNIKAS Martin. Restangular [en ligne]. , 2013 Disponible sur : https://github.com/mgonto/restangular (consulté en Avril 2014).

[GRUBER93] GRUBER Thomas R.. A translation approach to portable ontology specifications. **In** : Knowledge Acquisition. Vol. 5. , Avril 1993, 199-220 p.

[GUARINO95] GUARINO Nicola et GIARETTA Pierdaniele. Ontologies and knowledge bases: Towards a terminological clarification. **In** : Towards very Large Knowledge bases: Knowledge Building and Knowledge sharing. , 1995, 25-32 p.

[GUERAUD04] GUERAUD Viviane , ADAM Jean-Michel , PERNIN Jean-Philippe et al. L'exploitation d'Objets Pédagogiques Interactifs à distance : le projet FORMID. Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation [en ligne]. 2004, vol. 11. Disponible sur : http://sticef.univ-lemans.fr/ (consulté en Avril 2013).

[GUIZANI12] GUIZANI Nachoua , "Méta-modélisation de scénarios d'apprentissage en relation avec l'organisation des connaissances mobilisées," Université Joseph Fourier, Grenoble, Mémoire de Master Informatique Mathématique 2012.

[H2 Database14] H2Database. H2 Database [en ligne]. Disponible sur : http://h2database.com/html/performance.html (consulté en Avril 2014).

[HADJIAT11] HADJIAT Souad. HTML5 et les WebSockets [en ligne]. , Mars 2011 Disponible sur : http://blog.zenika.com/index.php?post/2011/02/25/Html5-et-les-webSockets (consulté en Avril 2014).

[HASHARBOT03] HASHARBOT. Framework [en ligne]. , 2003 Disponible sur : http://fr.wikipedia.org/wiki/Framework (consulté en Avril 2014).

[JQuery14] The JQuery Foundation. JQuery [en ligne]. Disponible sur : http://jquery.com/ (consulté en Avril 2014).

[jsTree14] BOZHANOV Ivan. jsTree [en ligne]. Disponible sur : http://www.jstree.com/ (consulté en Avril 2014).

[KALLENBORN14] KALLENBORN Gilbert. Sécurité [en ligne]. , Janvier 2014 Disponible sur : http://www.01net.com/editorial/611870/oracle-publie-demain-un-megapatch-pour-144-failles-dont-36-relatives-a-java/ (consulté en Avril 2014).

[LIG14] Laboratoire Informatique de Grenoble. LIG [en ligne]. Disponible sur : www.liglab.fr (consulté en Avril 2014).

[Love Bootstrap14] LoveBootstrap. Love Bootstrap [en ligne]. Disponible sur : http://lovebootstrap.com/ (consulté en Avril 2014).

[MANDRAN14] MANDRAN Nadine , "Présentation UnderTracks," Présentation 2014.

[MAPLE13] MAPLE Simon , SHELAJEV Oleg , MUUGA Sigmar et al. The Great Java Application Server Debate with Tomcat, JBoss, GlassFish, Jetty and Liberty Profile [en ligne]. , 2013 Disponible sur : http://zeroturnaround.com/rebellabs/the-great-java-application-server-

debate-with-tomcat-jboss-glassfish-jetty-and-liberty-profile (consulté en Avril 2014).

[MARTEL06] MARTEL Christian , VIGNOLLET Laurence , FERRARIS Christine et al., "Modeling collaborative learning activities on e-learning platforms," in *ICALT 2006 - Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies*, Kerkrade, The Netherlands, 2006, pp. 838-844.

[Maven14] Apache. Maven [en ligne]. Disponible sur : http://maven.apache.org/ (consulté en Avril 2014).

[NICOLA13] NICOLA Thierry. AngularJS: My solution to the ng-repeat performance problem [en ligne]. , 2013 Disponible sur : http://www.williambrownstreet.net/blog/2013/07/angularjs-my-solution-to-the-ng-repeat-performance-problem/ (consulté en Avril 2014).

[noe-kaleidoscope14] Kaléidoscope. noe-kaleidoscope [en ligne]. Disponible sur : http://www.noe-kaleidoscope.org/pub/about/ (consulté en Avril 2014).

[Oracle] Oracle. About [en ligne]. Disponible sur : http://www.java.com/fr/about/ (consulté en Avril 2014).

[Oracle14] Oracle. Release Dates [en ligne]. , 2014 Disponible sur : http://www.java.com/fr/download/faq/release_dates.xml (consulté en Avril 2014).

[PESTEL09] PESTEL Aaron. Running JBoss on port 80 or 443 [en ligne]. , 2009 Disponible sur : https://community.jboss.org/wiki/RunningJBossOnPort80Or443?_sscc=t (consulté en Avril 2014).

[PORTO13] PORTO Sebastian. A Comparison of Angular, Backbone, CanJS and Ember [en ligne]. , Avril 2013 Disponible sur : http://sporto.github.io/blog/2013/04/12/comparison-angular-backbone-can-ember/ (consulté en Avril 2014).

[PSYCHE03] PSYCHE Valéry , MENDES Olavo , et BOURDEAU Jacqueline. Apport de l'ingénierie ontologique aux environnements de formation à distance. Revue des Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation [en ligne]. 2003, vol. 10, pp. 89-126. Disponible sur : http://sticef.univ-lemans.fr/ (consulté en Avril 2013).

[Putty14] TATHAM Simon. Putty [en ligne]. Disponible sur : http://www.putty.org/ (consulté en Avril 2014).

[RESTEasy14] JBoss. RESTEasy [en ligne]. Disponible sur : http://www.jboss.org/resteasy (consulté en Avril 2014).

[SILENCE13] SILENCE Deep. War (Format de fichier) [en ligne]. , 2013 Disponible sur : http://fr.wikipedia.org/wiki/WAR_(format_de_fichier) (consulté en Avril 2014).

[SOUBOK06] SOUBOK. JavaScript Object Notation [en ligne]. , 2006 Disponible sur : http://fr.wikipedia.org/wiki/JavaScript_Object_Notation (consulté en Avril 2014).

[TIT_TOINOU13] TIT_TOINOU. JQuery, écrivez moins pour en faire plus [en ligne]. , 2013 Disponible sur : http://fr.openclassrooms.com/informatique/cours/jquery-ecrivez-moins-pour-faire-plus (consulté en Avril 2014).

[TPElec14] ADAM Jean-Michel. TPElec [en ligne]. Disponible sur : http://tpelec.imag.fr/tpelec.htm (consulté en Avril 2014).

[Undertracks14] MeTAH. Undertracks [en ligne]. Disponible sur : https://undertracks.imag.fr/ (consulté en Avril 2014).

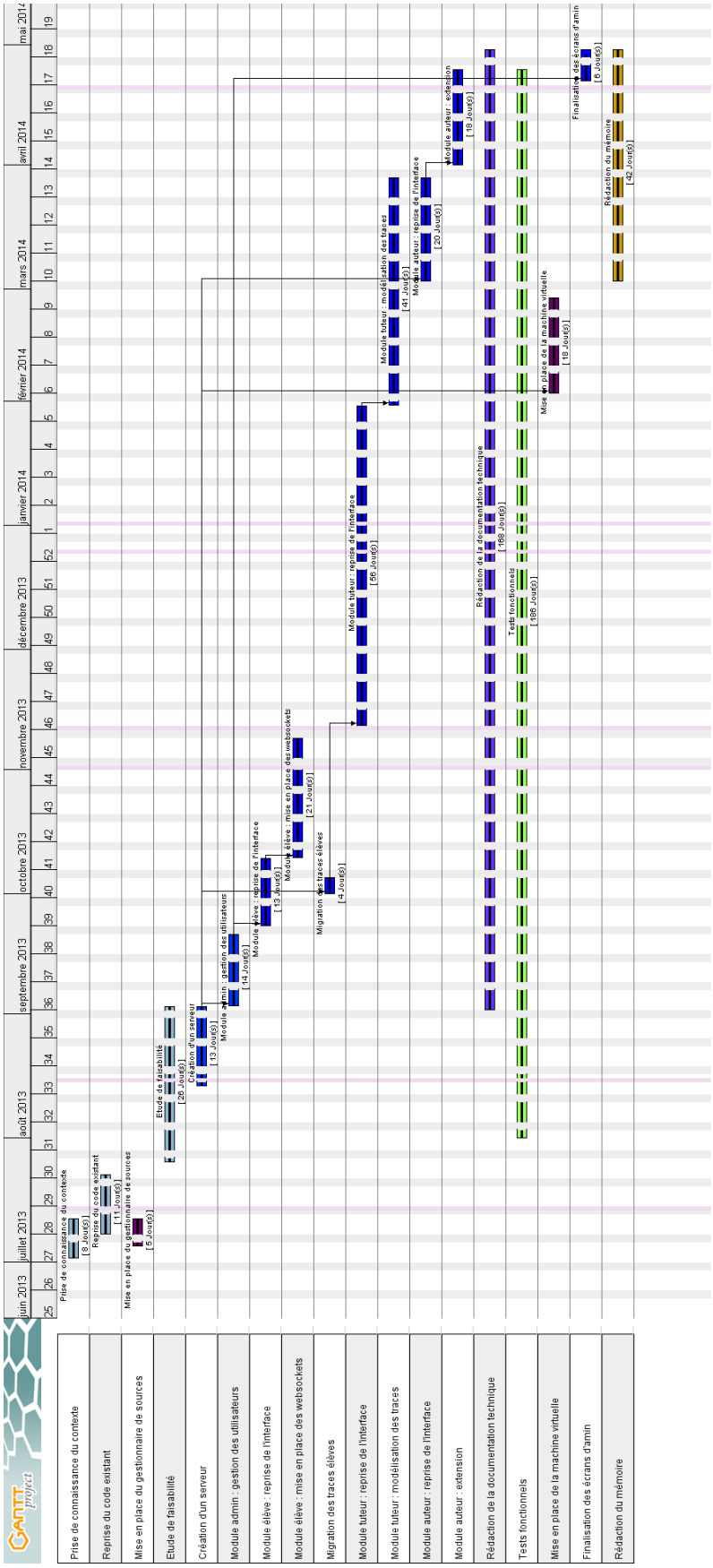
[Veille technologique 14] DURAND Alexandre , DESCHAMPS Mathias , et HOURLIER Thomas. Veille technologique [en ligne]. Disponible sur : http://cnode.fr/veilletechno/index.html (consulté en Avril 2014).

[Wikipedix13] Wikipedix. Applet [en ligne]. , Mars 2013 Disponible sur : http://fr.wikipedia.org/wiki/Applet (consulté en Avril 2014).

[WildFly14] JBoss. WildFly [en ligne]. Disponible sur : http://wildfly.org/ (consulté en Avril 2014).

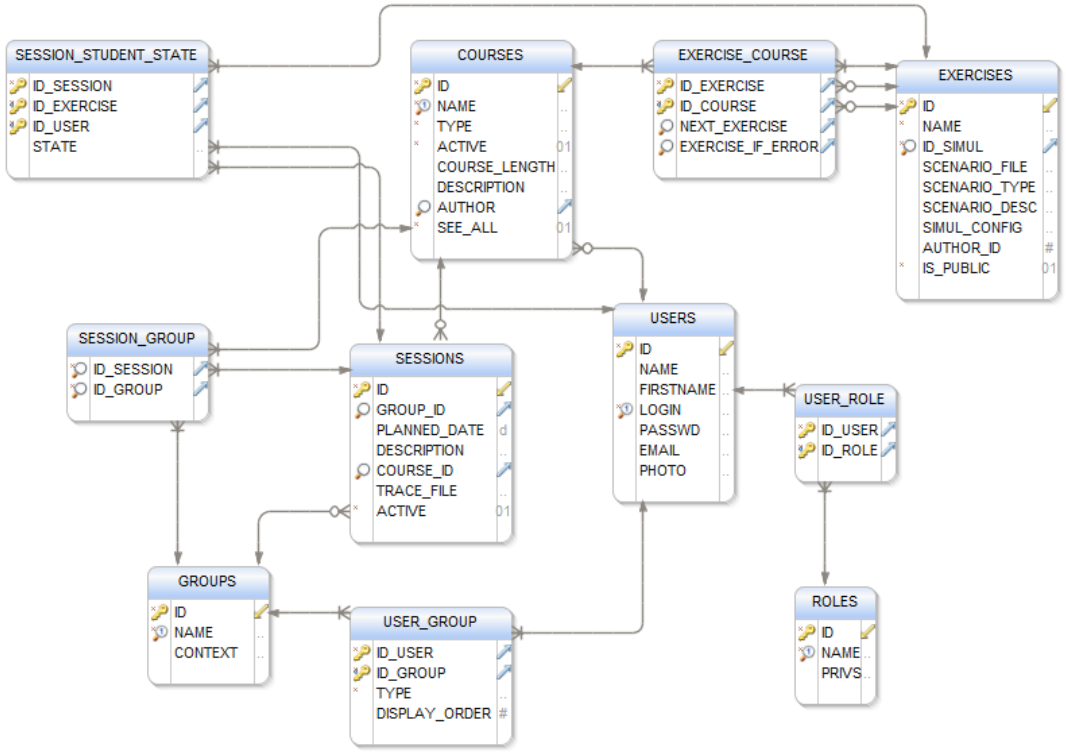
# Annexes

## A. Diagramme de Gantt

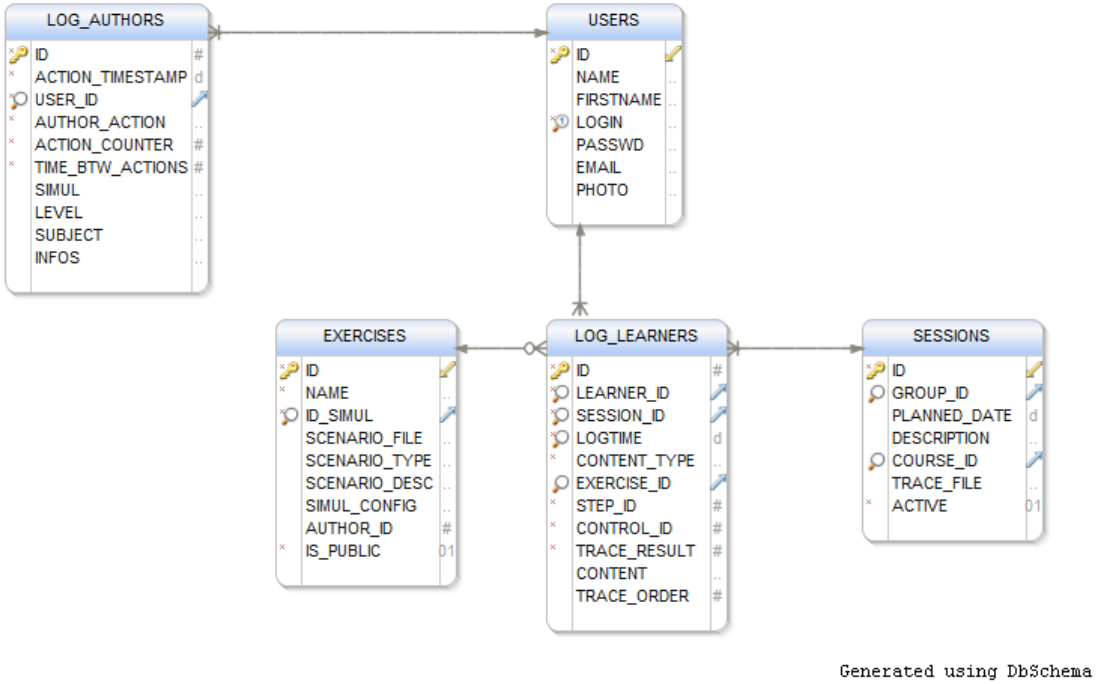


## B. Diagramme complet de base de données

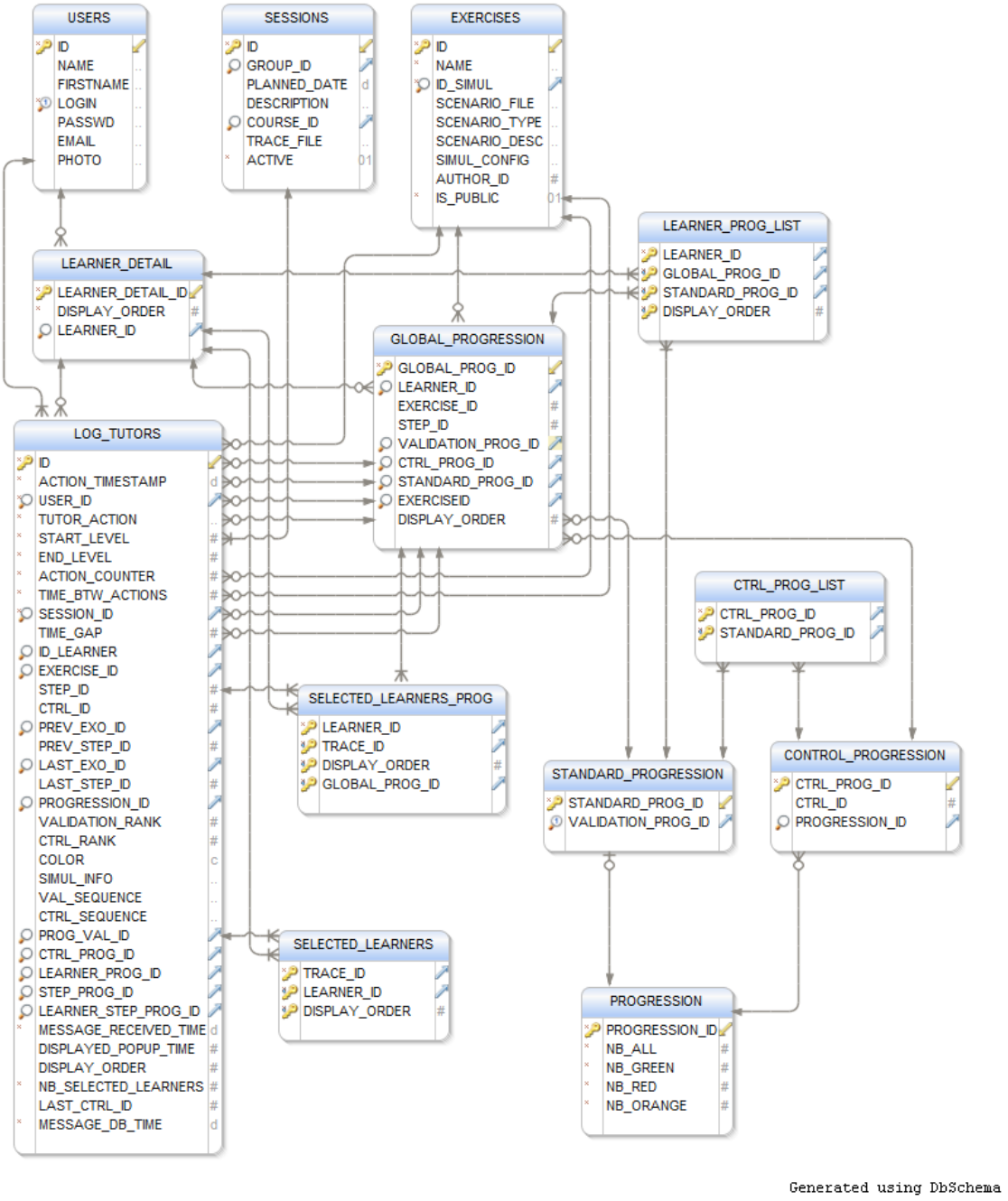
### 1. Gestion des utilisateurs et des séances de cours



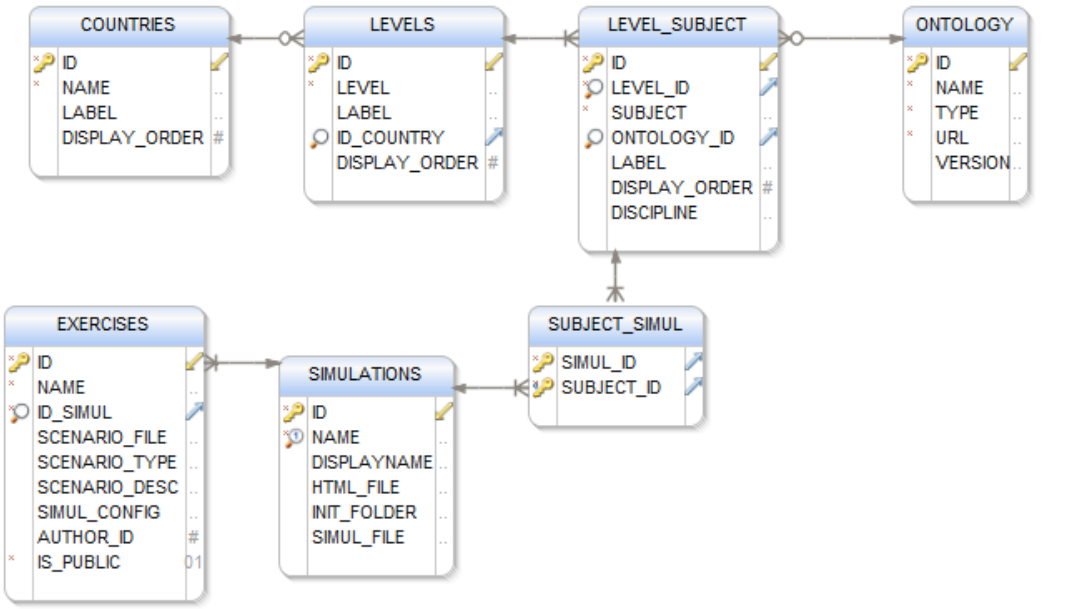

### 2. Gestion des traces élève et auteur



## 3. Gestion des traces tuteur

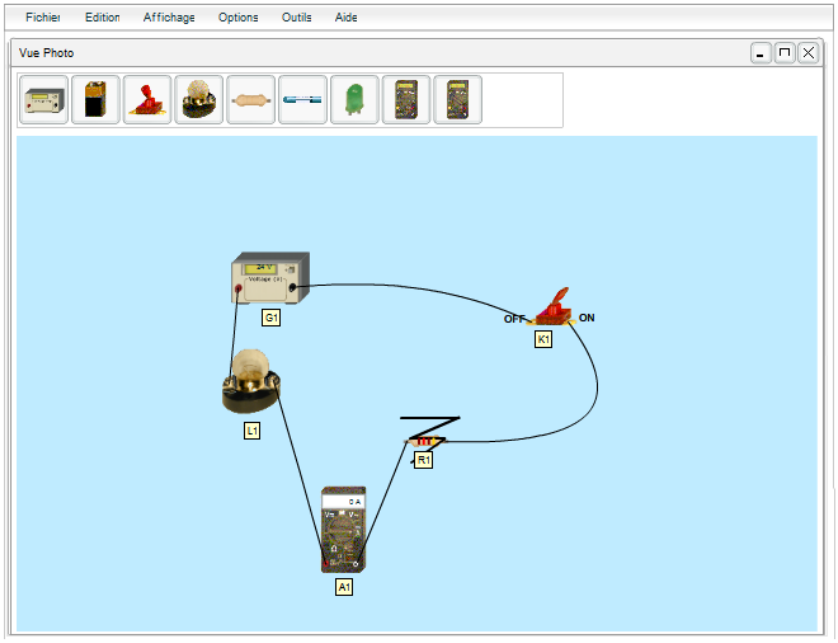


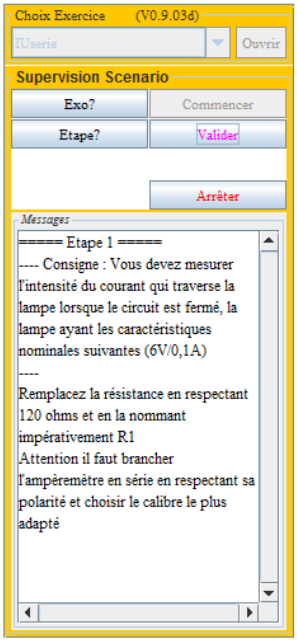
## 4. Gestion de la hiérarchie de l'ontologie



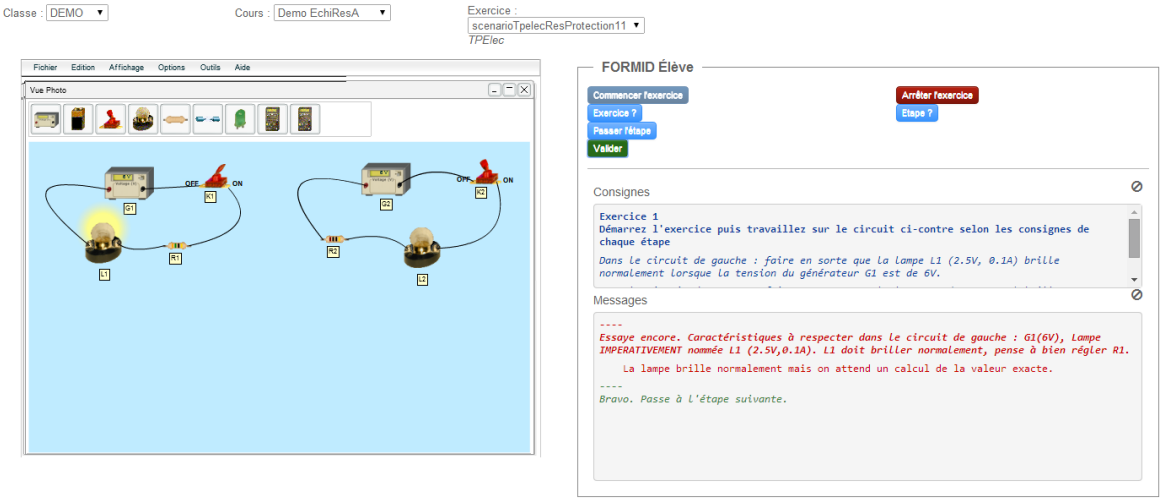
## C. Écrans de la suite FORMID avant et après transformation

### Ancienne version : écran module élève

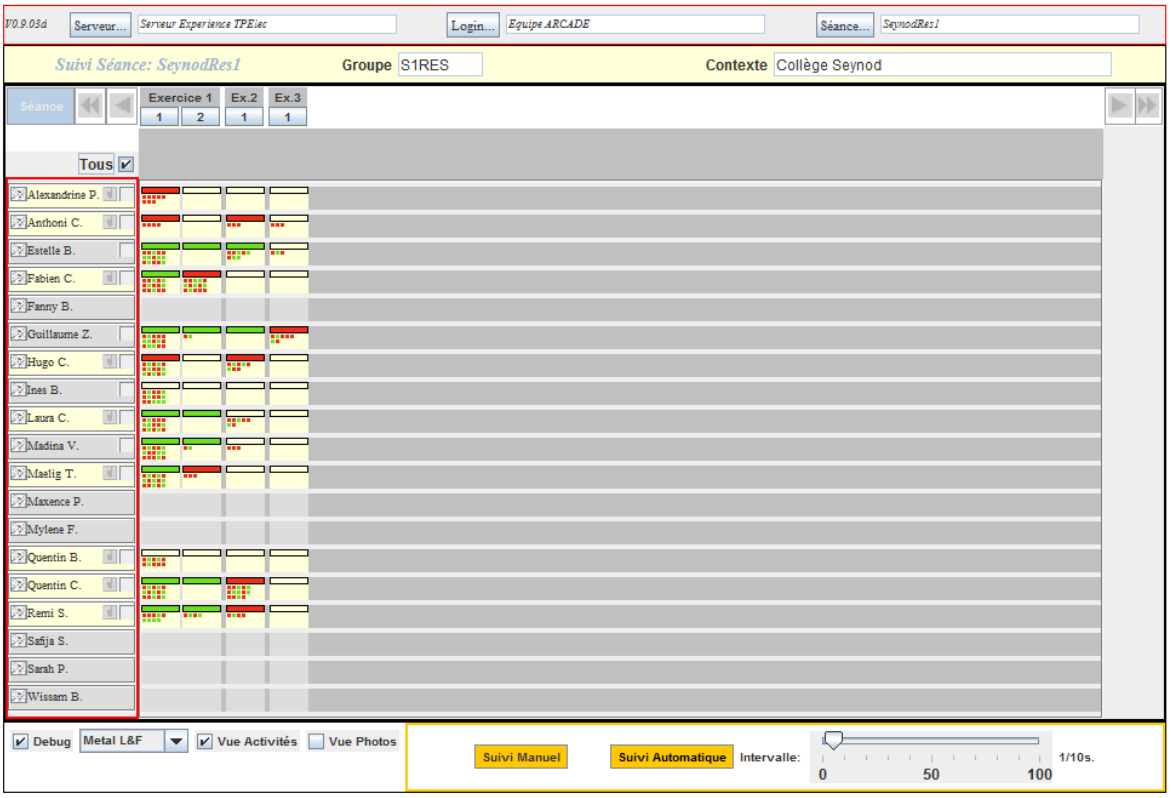


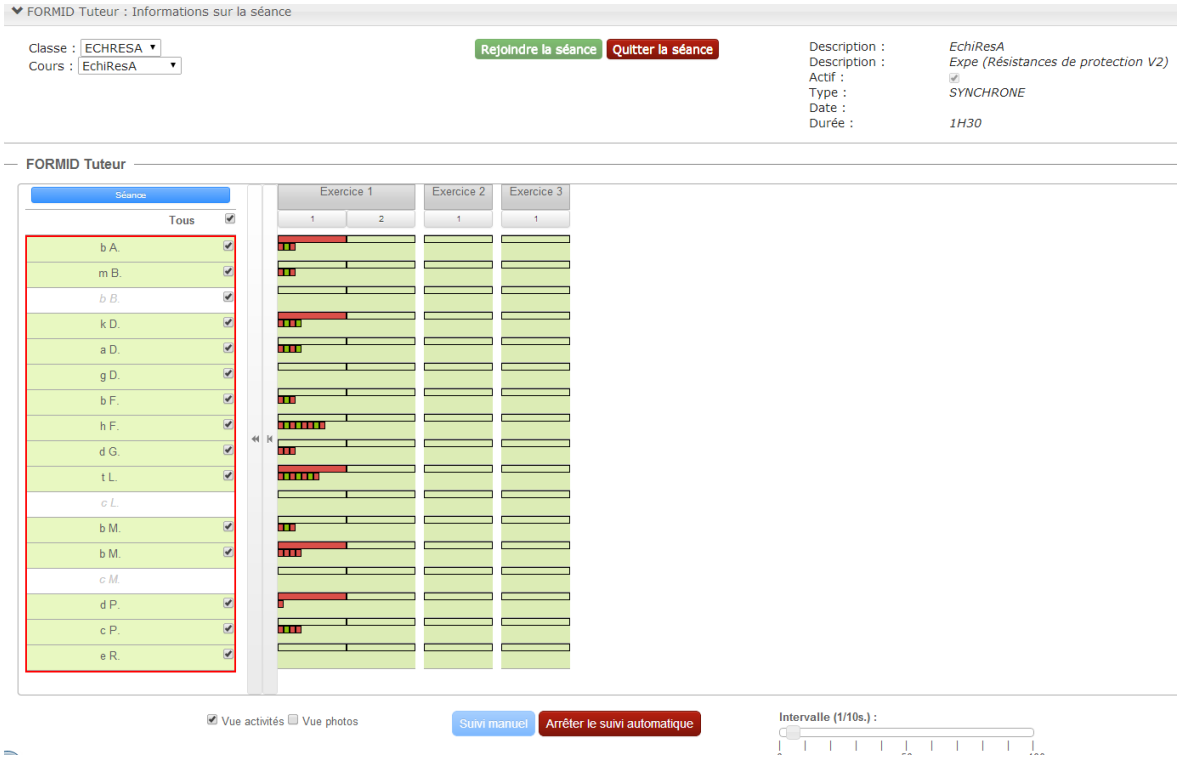

### Nouvelle version : écran module élève



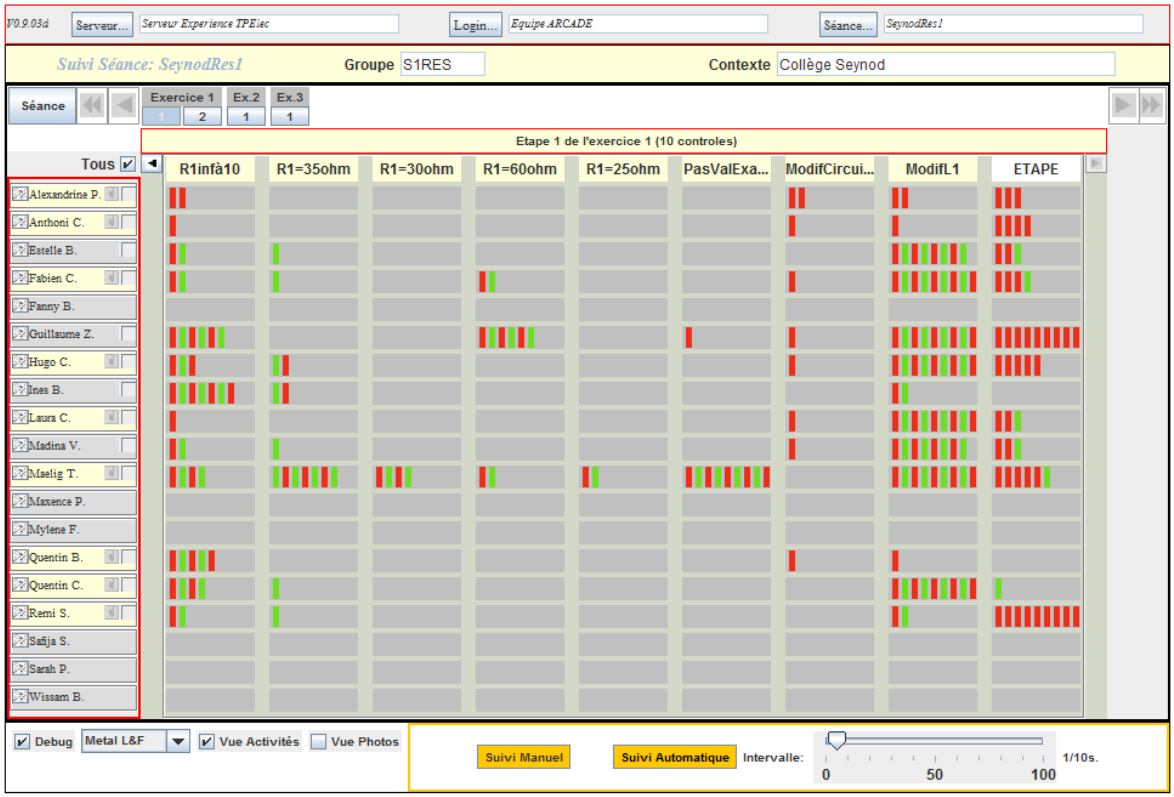
## Ancienne version : écran module tuteur – suivi global



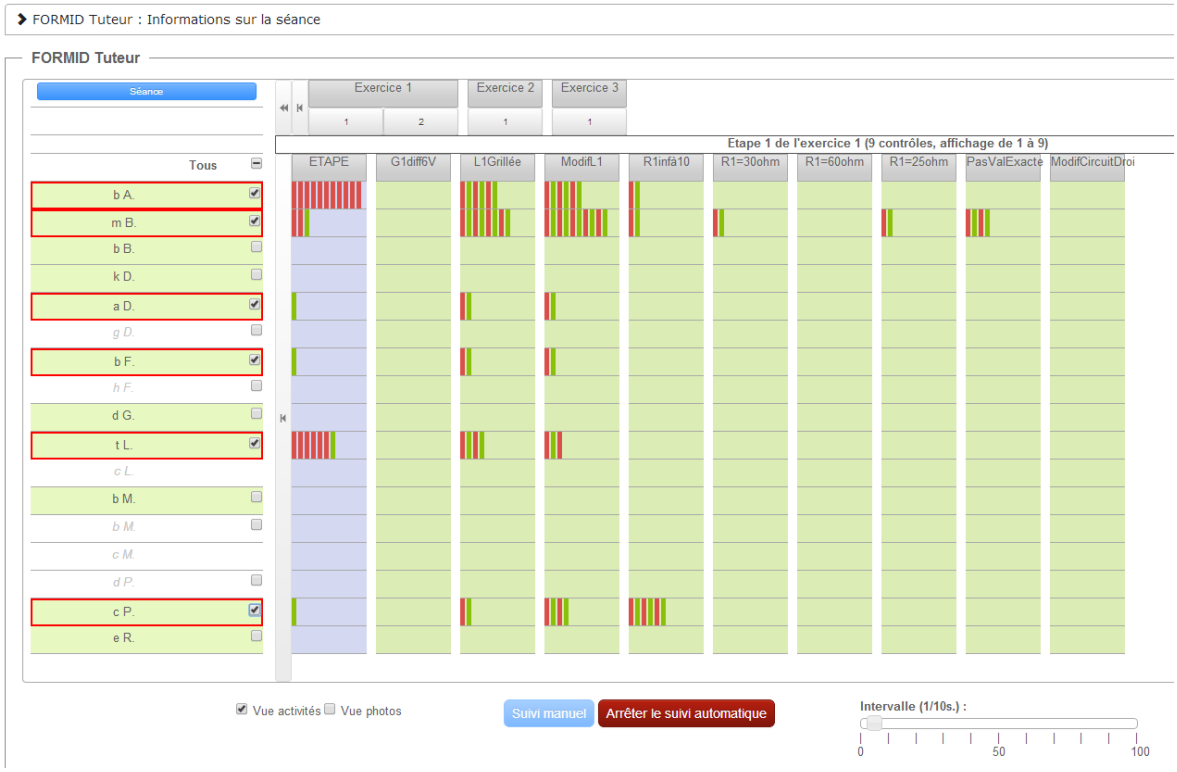

## Nouvelle version : écran module tuteur – suivi global



## Ancienne version : écran module tuteur – suivi étape

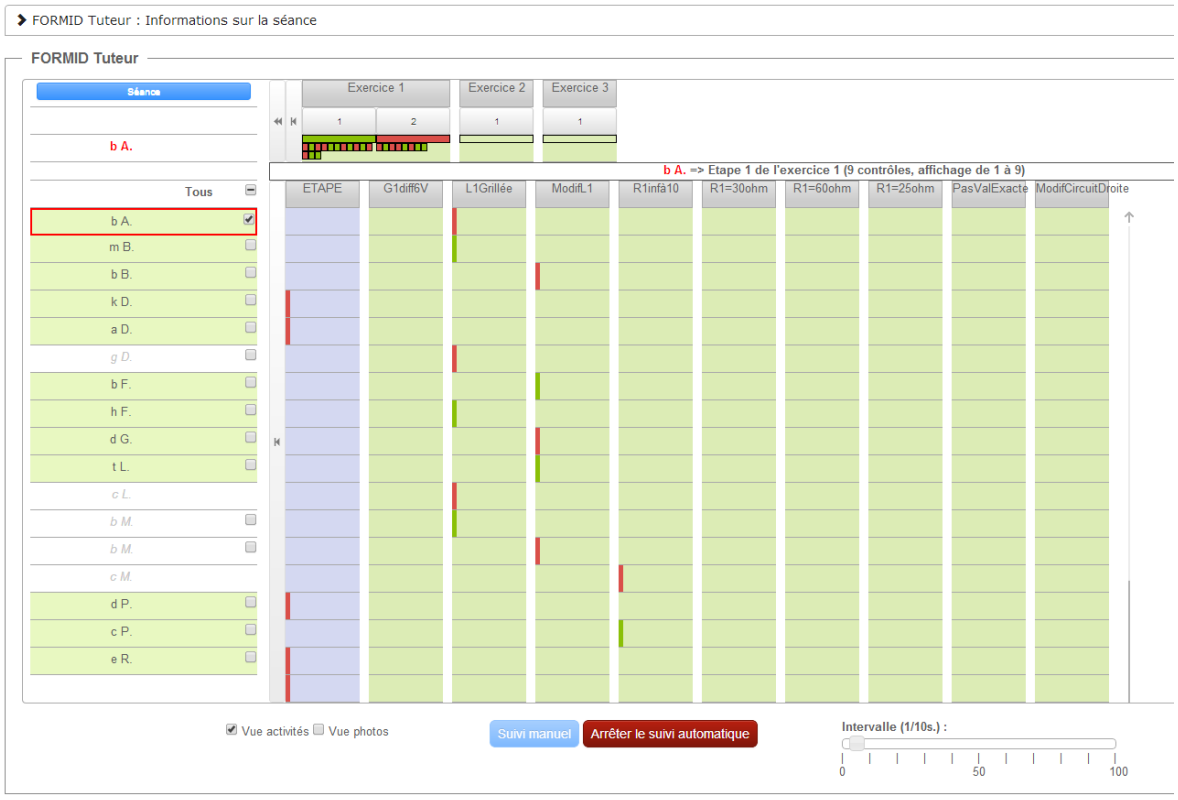



## Nouvelle version : écran module tuteur – suivi étape

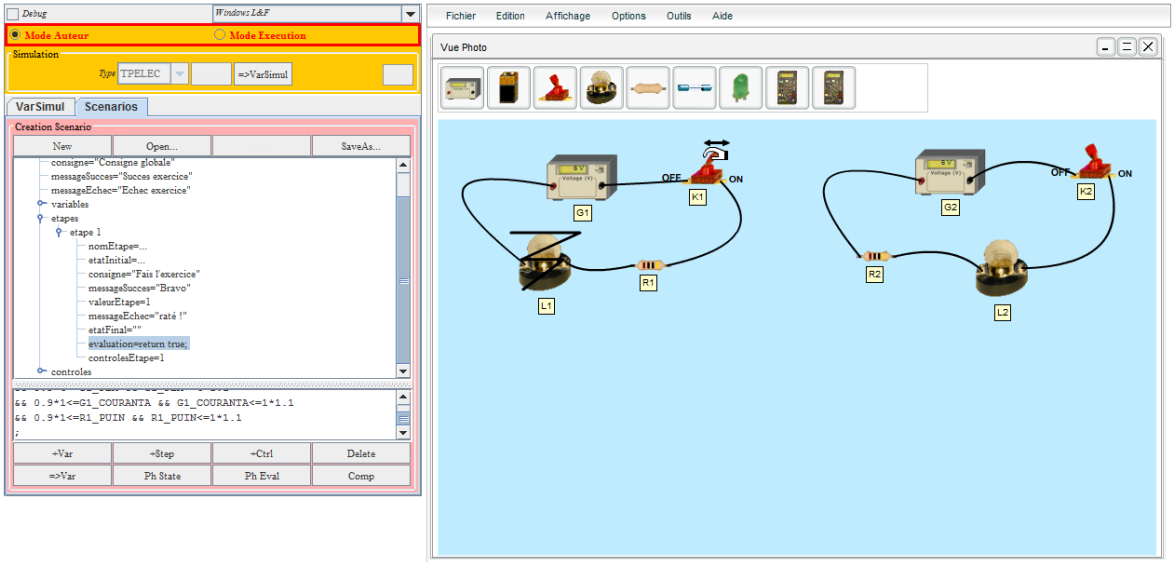


## Ancienne version : écran module tuteur – suivi élève

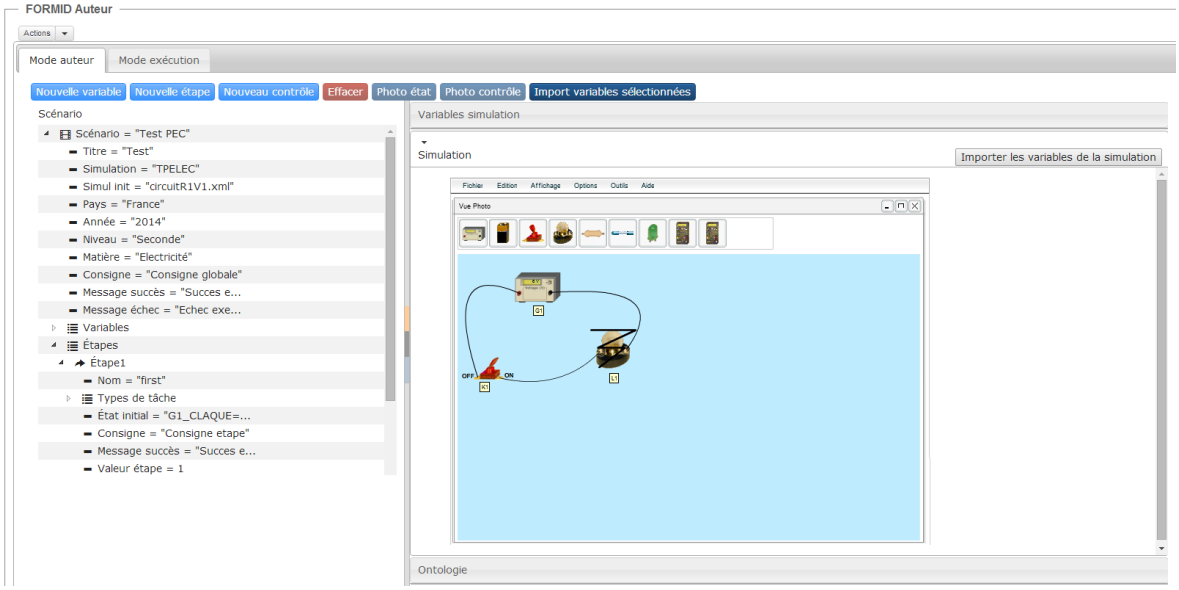

## Nouvelle version : écran module tuteur – suivi élève



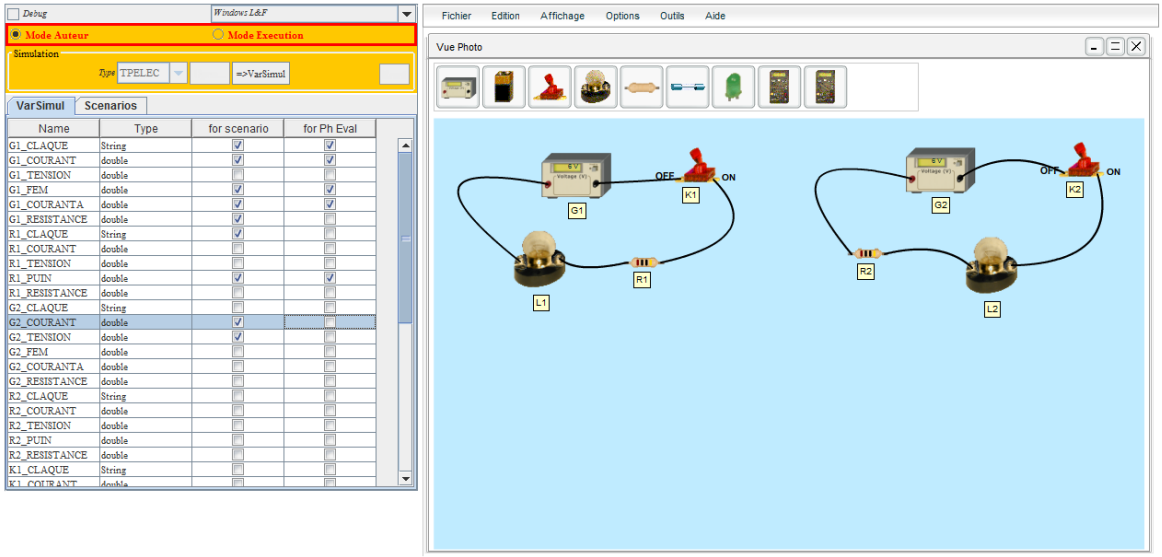
## Ancienne version : écran module auteur – arbre / simulation



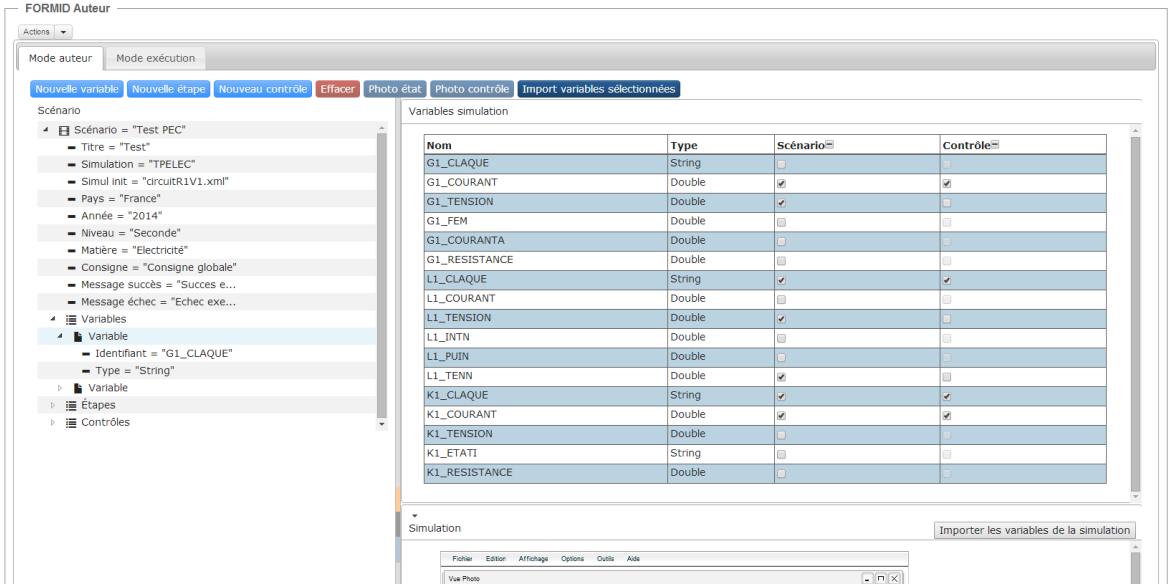

## Nouvelle version : écran module auteur – arbre / simulation



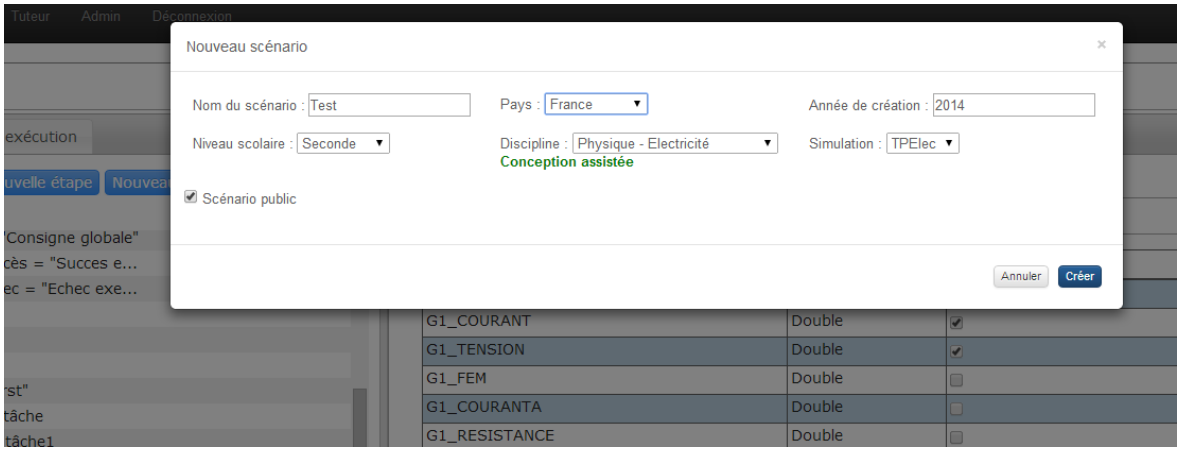
## Ancienne version : écran module auteur – choix des variables simulation



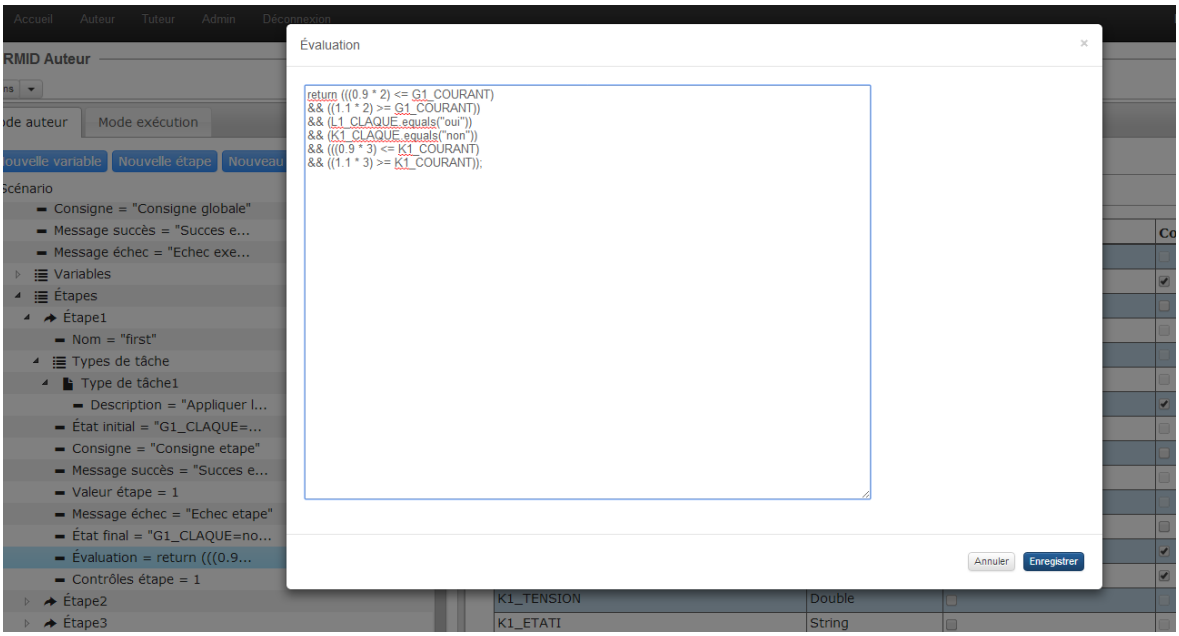

## Nouvelle version : écran module auteur – choix des variables simulation




## Nouvelle version : écran module auteur – création de scénario

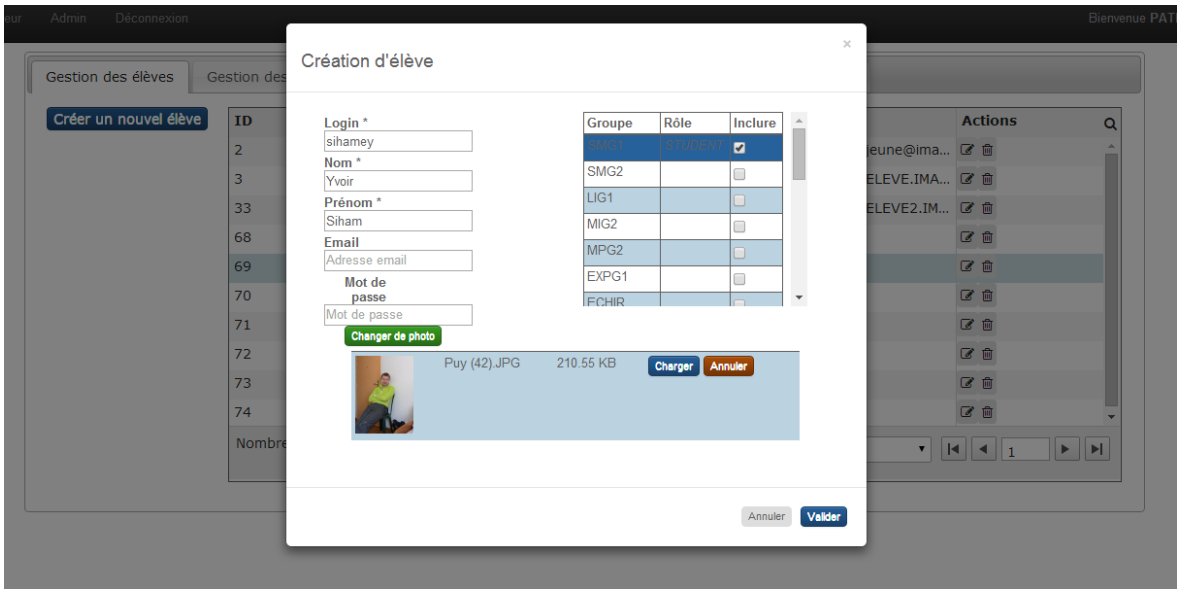


## Nouvelle version : écran module auteur – Edition de condition de validation

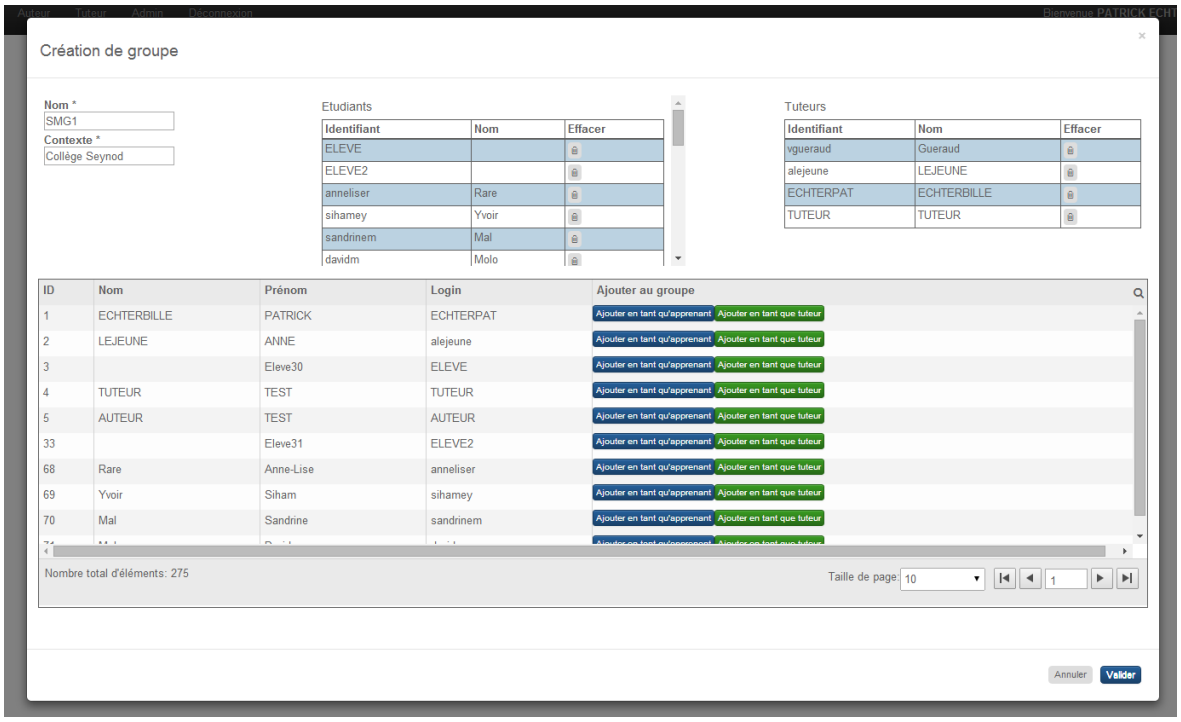



## Nouvelle version : écran module auteur – Aide à la conception

## Nouvelle version : écran module administration – Gestion des élèves




## Nouvelle version : écran module administration – Gestion des groupes



## D. Exemple de définition de scénario

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario nom="Test PEC">
  <titre>Test</titre>
  <simulation>TPELEC</simulation>
  <initFile>circuitR1V1.xml</initFile>
  <pays>France</pays>
  <annee>2014</annee>
  <niveauClasse>Seconde</niveauClasse>
  <matiere>Electricité</matiere>
  <consigne>Consigne globale</consigne>
  <messageSucces>Succes exercice</messageSucces>
  <messageEchec>Echec exercice</messageEchec>
  <variables>
    <variable>
      <ident>G1_CLAQUE</ident>
      <type>String</type>
    </variable>
    <variable>
      <ident>G1_COURANT</ident>
      <type>Double</type>
    </variable>
  </variables>
  <etapes>
    <etape>
      <nomEtape>first</nomEtape>
      <consigne>Consigne etape</consigne>
      <valeurEtape>1</valeurEtape>
<etatInitial>G1_CLAQUE=non;G1_COURANT=0;G1_TENSION=0;G1_FEM=6;G1_COURAN
TA=1;G1_RESISTANCE=0.1;L1_CLAQUE=oui;L1_COURANT=0;L1_TENSION=0;L1_INTN=
0.1;L1_PUIN=0.6;L1_TENN=6;K1_CLAQUE=non;K1_COURANT=0;K1_TENSION=0;K1_ET
ATI=oui;K1_RESISTANCE=0.1;</etatInitial>
<etatFinal>G1_CLAQUE=non;G1_COURANT=0;G1_TENSION=0;G1_FEM=6;G1_COURANTA
=1;G1_RESISTANCE=0.1;L1_CLAQUE=oui;L1_COURANT=0;L1_TENSION=0;L1_INTN=0.
1;L1_PUIN=0.6;L1_TENN=6;K1_CLAQUE=non;K1_COURANT=0;K1_TENSION=0;K1_ETAT
I=oui;K1_RESISTANCE=0.1;</etatFinal>
      <evaluation>return (G1_CLAQUE.equals("non")) &amp;amp;&amp;amp;
(G1_COURANT.equals(0));</evaluation>
      <messageSucces>Succes etape</messageSucces>
      <messageEchec>Echec etape</messageEchec>
      <controlesEtape>1</controlesEtape>
      <typeTaches>
        <typeTache>
          <description>Appliquer la loi d'OHM aux bornes de
R</description>
        </typeTache>
      </typeTaches>
    </etape>
    <etape>
      <nomEtape />
      <consigne>Consigne étape</consigne>
      <valeurEtape>1</valeurEtape>
      <etatInitial />
      <etatFinal />
      <evaluation>return (G1_CLAQUE.equals("non")) &amp;amp;&amp;amp;
(G1_COURANT.equals(2));</evaluation>
      <messageSucces>Succès étape</messageSucces>
      <messageEchec>Échec étape</messageEchec>
      <controlesEtape />
      <typeTaches>
        <typeTache>
```

```
          <description>Calculer la tension aux bornes de R dans un
circuit en série…</description>
          <savoirs>
            <savoir>
              <label>voltageAdditivityInSeriesCircuits</label>
              <description>La loi d'additivité tensions dans un circuit
en série (La tension aux bornes du générateur est égale à la somme des
tensions aux bornes des dipôles récepteurs)</description>
            </savoir>
          </savoirs>
          <resolutionAttendue>
            <tache>
              <description>Appliquer la loi d’additivité des tensions
dans un circuit en série</description>
            </tache>
            <tache>
              <description>Calculer la grandeur cherchée à partir de
l’égalité obtenue</description>
            </tache>
          </resolutionAttendue>
          <erreursTypiques>
            <tache>
              <description>techtho1b</description>
            </tache>
          </erreursTypiques>
        </typeTache>
      </typeTaches>
    </etape>
  </etapes>
  <controles>
    <controle>
      <id />
      <nom>TitreCtrl1</nom>
      <messageTrue>Detection Controle</messageTrue>
      <valeurTrue>-1</valeurTrue>
      <messageFalse>Fin Detection</messageFalse>
      <valeurFalse>1</valeurFalse>
      <evaluation>return (G1_CLAQUE.equals("tata")) &amp;amp;&amp;amp;
(G1_COURANT.equals(0));</evaluation>
      <terminal>false</terminal>
    </controle>
    <controle>
      <id />
      <nom>ctrl2</nom>
      <messageTrue>Détection ctrl</messageTrue>
      <valeurTrue>-1</valeurTrue>
      <messageFalse>Fin ctrl</messageFalse>
      <valeurFalse>1</valeurFalse>
      <evaluation>return (G1_CLAQUE.equals("gni")) &amp;amp;&amp;amp;
(G1_COURANT.equals(5));</evaluation>
      <terminal>false</terminal>
      <erreurTypiques>
        <tache>
          <description>techtho1b</description>
        </tache>
      </erreurTypiques>
    </controle>
  </controles>
</scenario>
```

## E. Exemple de code généré par le scénario

```
//Code produit "automatiquement" par FormidScenario_xmlToJava2.xsl (v1)
package scenarioTpelecResProtection1;

import fr.imag.lig.metah.formidweb.formidScenario.*;
import
fr.imag.lig.metah.formidweb.formidScenario.scenarioFactory.ScenarioCrea
torDefInterface;

public class ScenarioCreatorDef implements ScenarioCreatorDefInterface
{

    public Scenario createScenario() {

        return new Scenario() {
//debut vraie def du scenario-----------------------------------------
---
            double L1_COURANT;
            double L2_COURANT;
            String L1_CLAQUE;
            String L2_CLAQUE;
            double G1_FEM;
            double G2_FEM;
            double L1_INTN;
            double L2_INTN;
            double L1_TENN;
            double L2_TENN;
            double R1_RESISTANCE;
            double R2_RESISTANCE;

            public void updateCurrentState() {

                L1_COURANT = _getVarDouble("L1_COURANT");
                L2_COURANT = _getVarDouble("L2_COURANT");
                L1_CLAQUE = getValue("L1_CLAQUE");
                L2_CLAQUE = getValue("L2_CLAQUE");
                G1_FEM = _getVarDouble("G1_FEM");
                G2_FEM = _getVarDouble("G2_FEM");
                L1_INTN = _getVarDouble("L1_INTN");
                L2_INTN = _getVarDouble("L2_INTN");
                L1_TENN = _getVarDouble("L1_TENN");
                L2_TENN = _getVarDouble("L2_TENN");
                R1_RESISTANCE = _getVarDouble("R1_RESISTANCE");
                R2_RESISTANCE = _getVarDouble("R2_RESISTANCE");

            }//observe

            {//instance initializer block

                nom = "TpelecResProtection1";
                titre = "Scenario TPelec: Résistance de Protection 1";
                variablesModele = new String[]{
                    "L1_COURANT", "L2_COURANT", "L1_CLAQUE",
"L2_CLAQUE", "G1_FEM", "G2_FEM", "L1_INTN", "L2_INTN", "L1_TENN",
"L2_TENN", "R1_RESISTANCE", "R2_RESISTANCE"
                };
                consigneGlobale = "Exercice 1 \nDémarrez l'exercice
puis travaillez sur le circuit ci-contre selon les consignes de chaque
étape";
                messageSucces = "Exercice réussi. Bon travail !  ";
                messageEchec = "Exercice non réussi.";
```

```
                nbEtapes = 2;
                etapes = new Etape[nbEtapes];
                nbControles = 19;
                controles = new Controle[nbControles];

                etapes[0] = new Etape(
                        "Dans le circuit de gauche : faire en sorte que
la lampe L1 (2.5V, 0.1A) brille normalement lorsque la tension du
générateur G1 est de 6V. ", "----Bravo. Passe à l'étape suivante.", 1,
"Essaye encore. Caractéristiques à respecter dans le circuit de gauche
: G1(6V), L1 (2.5V,0.1A).", new
PhotoEtat("L1_INTN=0.1;L2_INTN=0.1;L1_TENN=2.5;L2_TENN=2.5;G1_FEM=6;G2_
FEM=6;L1_CLAQUE=non;L2_CLAQUE=non;R1_RESISTANCE=10;R2_RESISTANCE=10;")/
/etat initial
                        , new
PhotoEtat("L1_INTN=0.1;L2_INTN=0.1;L1_TENN=2.5;L2_TENN=2.5;G1_FEM=6;G2_
FEM=6;L1_CLAQUE=non;L2_CLAQUE=non;R1_RESISTANCE=35;R2_RESISTANCE=10;")/
/etat final
                        , new Controle() {//fin d'etape
                            public boolean evalCond() {

                                return (5.9 < G1_FEM && G1_FEM < 6.1
                                && L1_CLAQUE.equals("non")
                                && 0.09 < L1_INTN && L1_INTN < 0.11
                                && 2.4 < L1_TENN && L1_TENN < 2.6
                                && 34.9 < R1_RESISTANCE &&
R1_RESISTANCE < 35.1
                                && 0.0995 < Math.abs(L1_COURANT) &&
Math.abs(L1_COURANT) < 0.0997);
                            }
                        }, new int[]{1, 2, 3, 4, 5, 6, 7, 8, 9, 18}
                );

                etapes[1] = new Etape(
                        "Dans le circuit de droite : faire en sorte que
la lampe L2 (2.5V, 0.1A) brille normalement lorsque la tension du
générateur G2 est de 6V.", "Bravo.", 1, "Essaye encore.
Caractéristiques à respecter dans le circuit de droite : G2(6V)", new
PhotoEtat("L2_INTN=0.1;L2_TENN=2.5;G2_FEM=6;L2_CLAQUE=non;R2_RESISTANCE
=10;) , new
PhotoEtat("L2_INTN=0.1;L2_TENN=2.5;G2_FEM=6;L2_CLAQUE=non;R2_RESISTANCE
=35;")//etat final
                        , new Controle() {//fin d'etape
                            public boolean evalCond() {
                                return (5.9 < G2_FEM && G2_FEM < 6.1
                                && L2_CLAQUE.equals("non")
                                && 0.09 < L2_INTN && L2_INTN < 0.11
                                && 2.4 < L2_TENN && L2_TENN < 2.6
                                && 34.9 < R2_RESISTANCE &&
R2_RESISTANCE < 35.1
                                && 0.0995 < Math.abs(L2_COURANT) &&
Math.abs(L2_COURANT) < 0.0997);
                            }
                        }, new int[]{10, 11, 12, 13, 14, 15, 16, 17,
19}
                );
                controles[0] = new Controle(
                        "G1diff6V", false, -1, 1
                ) {
                    public String messageTrue() {
```

```
                        return "Respecte les consignes, la tension du
générateur ne doit pas être modifiée : remets-la à 6V.";
                    }

                    public String messageFalse() {
                        return "null";
                    }

                    public boolean evalCond() {
                        return (G1_FEM < 5.9 || 6.1 < G1_FEM);

                    }
                };
                controles[1] = new Controle(
                        "L1Grillée", false, -1, 1
                ) {
                    public String messageTrue() {
                        return "Tu dois changer la lampe que tu viens
de faire griller. Par sécurité, ouvre l'interrupteur avant de la
changer.";
                    }
                    public String messageFalse() {
                        return "null";
                    }
                    public boolean evalCond() {

                        return L1_CLAQUE.equals("oui");

                    }
                };
                controles[2] = new Controle(
                        "R1infà10", false, -1, 1
                ) {
                    public String messageTrue() {
                        return "Réfléchis à l'influence de la valeur de
la résistance sur l'intensité du courant dans le circuit.";
                    }
                    public String messageFalse() {
                        return "null";
                    }
                    public boolean evalCond() {
                        return (R1_RESISTANCE < 10.0);

                    }
                };
                controles[3] = new Controle(
                        "R1=35ohm", false, 1, -1
                ) {
                    public String messageTrue() {
                        return "null";
                    }
                    public String messageFalse() {
                        return "null";
                    }
                    public boolean evalCond() {
                        return (34.9 < R1_RESISTANCE && R1_RESISTANCE <
35.1);

                    }
                };
```

```
            }//end of instance initializer block
//Fin def du scenario  ------------------------------------------------
--
        };
    }//createScenario
}//class
```

## F. Code source de la liaison websocket

### 1. Côté serveur

```
package fr.imag.lig.metah.formidweb.WebSocket;

import fr.imag.lig.metah.formidweb.Utils.FormidLog;
import fr.imag.lig.metah.formidweb.formidScenario.FormIDApplication;
import java.io.IOException;
import org.atmosphere.config.service.Disconnect;
import org.atmosphere.config.service.ManagedService;
import org.atmosphere.config.service.Message;
import org.atmosphere.config.service.Ready;
import org.atmosphere.cpr.AtmosphereResource;
import org.atmosphere.cpr.AtmosphereResourceEvent;
import static org.atmosphere.cpr.AtmosphereResourceFactory.getDefault;
import org.atmosphere.cpr.BroadcasterFactory;

/**
 * Creation date : 23 août 2013
 *
 * Classe implémentant le canal de dialogue entre l'élève et le
moniteur
 * de scénario
 *
 * C'est un canal double sens, à l'initiative du serveur ou de l'élève
 *
 * Cela n'est possible en HTML (actuellement) qu'avec la technologie
des
 * websockets
 *
 * Utilisation de la librairie Atmosphere
 * ({@link https://github.com/Atmosphere}) pour garantir un mode simulé
 * de communication à double-sens pour les navigateurs ne supportant
pas
 * les websockets (IE8 et antérieur)
 *
 * La librairie Atmosphère génère du code grâce aux annotations, d'où
la
 * simplicité de cette classe
 *
 * @author Patrick Echterbille
 */
//Le nom sera une chaine de caractères aléatoire générée par la classe
TokenUtils
@ManagedService(path = "{studentws: [a-zA-Z][a-zA-Z_0-9]*}")
public class StudentWS {

    private final static FormidLog Log =
FormidLog.getLogger(StudentWS.class);
    private String studentChannel;
    private BroadcasterFactory factory;
    private String mappedPath;

    /**
     *
```

```
     * Appelée lorsque la connexion a été établie et mise en attente,
     * càd prête  à recevoir des messages. Le client a déjà appelé une
     * servlet classique pour obtenir un canal disponible.
     *
     * L'annotation Ready configure la méthode pour répondre à
l'appelant
     * (BROADCASTER) et détermine la façon d'encoder l'objet retourné
     * (JsonEncoder)
     *
     * @param r La ressource contenant les informations nécessaires à
la
     * création du canal.
     * @return Le message donnant le résultat de la connexion
     */
    @Ready(value = Ready.DELIVER_TO.BROADCASTER, encoders =
{JsonEncoder.class})
    public MessageBean onReady(final AtmosphereResource r) {
        Log.info(r.transport() + " : Browser " + r.uuid() + "
connected. (Path: " + r.getBroadcaster().getID() + ")");
        if (studentChannel == null) {
            mappedPath = r.getBroadcaster().getID();
            // Get rid of the AtmosphereFramework mapped path.
            studentChannel = mappedPath.split("/")[2];
            factory = r.getAtmosphereConfig().getBroadcasterFactory();
        }
        MessageBean b = new MessageBean();
        b.setSessionId(studentChannel);
        b.setInError(Boolean.FALSE);
        b.setType("connection");
        b.setMessage("Connexion réussie");
        b.setUserUUID(r.uuid());
        return b;
    }

    /**
     *
     * Appelée lorsque le client se déconnecte, ou qu'une coupure
     * inattendue du réseau a lieu
     *
     * @param event Le type d'évènement provoquant la coupure
     */
    @Disconnect
    public void onDisconnect(AtmosphereResourceEvent event) {
        if (event.isCancelled()) {
            // We didn't get notified, so we remove the user.
            Log.info(event.getResource().transport() + " : Browser " +
event.getResource().uuid() + " unexpectedly disconnected");
        } else if (event.isClosedByClient()) {
            Log.info(event.getResource().transport() + " : Browser " +
event.getResource().uuid() + " closed the connection");
        }
    }

    /**
     *
     * Appelée lorsque l'élève envoie un message. Lecture du message et
     * redirection vers le service adéquat.
     *
     * L'annotation "decoders" permet de travailler directement sur un
     * objet Java (et non JSON)
     *
```

```
     * @param message Le message reçu
     * @throws IOException
     */
    @Message(decoders = {MessageBeanDecoder.class})
    public void onMessage(MessageBean message) throws IOException {
        //Recherche de l'identifiant de l'utilisateur sur le canal
        AtmosphereResource r =
getDefault().find(message.getUserUUID());

        //On analyse le type du message pour redirection vers le
service adéquat
        switch (message.getType()) {
            case "start":
                FormIDApplication.getInstance().startExo(this,
message);
                break;
            case "validate":
                FormIDApplication.getInstance().validate(message);
                break;
            case "exo":
                FormIDApplication.getInstance().getExoTip(message);
                break;
            case "step":
                FormIDApplication.getInstance().getStepTip(message);
                break;
            case "stop":
                FormIDApplication.getInstance().stopExo(message);
                break;
            case "state":

FormIDApplication.getInstance().sendSimulNotif(message);
                break;
            case "disconnect":
                FormIDApplication.getInstance().disconnect(message);
                break;
            default: //do nothing.
        }

    }

    /**
     *
     * Méthode d'envoi de message à l'initiative du serveur
L'annotation
     * "encoders" permet d'envoyer l'objet dans un format JSON
     *
     * @param msg Le message à envoyer
     */
    @Message(encoders = {JsonEncoder.class})
    public void sendMessage(MessageBean msg) {
        Log.debug("Incoming message: " + msg.getMessage() + " pour
channel " + msg.getSessionId());

        //On trouve l'utilisateur unique sur le canal
        AtmosphereResource r = getDefault().find(msg.getUserUUID());

        if (r != null) {
            Log.debug(r.transport() + " : Browser: " +
msg.getUserUUID() + " pour channel " + msg.getSessionId() + " with path
" + mappedPath);
            if (factory == null) {
```

```
                factory =
r.getAtmosphereConfig().getBroadcasterFactory();
            }
            //Envoi du message sur le canal à l'utilisateur sélectionné
            factory.lookup(mappedPath).broadcast(msg, r);
        }

        Log.info("Server has sent " + msg.getMessage() + " to " +
msg.getUserId());
    }
}
```

### 2. Côté client

```
angular.module('formidweb').factory('StudentService', ['$q',
'$rootScope', function($q, $rootScope) {
        // We return this object to anything injecting our service
        var Service = {};

        var socket = atmosphere;
        var subSocket = null;
        var output;
        var broadcastId;

        var userId;
        var exoId;
        var transport = 'websocket';
        var userUUID = 0;

        Service.init = function(user, b_id, exo, _courseId, resetFct) {
            if (subSocket) {
                userUUID = 0;
                socket.unsubscribe();
                subSocket = null;
            }
            oldState = {};
            initStudent();
            userId = user;
            exoId = exo;
            courseId = _courseId;
            broadcastId = b_id;
            reset = resetFct;
            // We are now ready to cut the request
            var request = {
                contentType: "application/json",
                url: "ws/" + broadcastId,
                logLevel: 'debug',
                transport: transport,
                //connectTimeout: 10000,
                trackMessageLength: true,
                fallbackTransport: 'long-polling',
                reconnectInterval: 5000,
                //maxReconnectOnClose: 3,
                //dropAtmosphereHeaders: false,
                timeout: 10 * 60 * 1000 // 10 minutes
            };

            request.onOpen = function(response) {
                transport = response.transport;
                logServer('Session ' + broadcastId + ' with transport :
' + transport, "WEBSOCKET MODULE");
```

```
                userUUID = response.request.uuid;
                Service.startExercice(userId, exoId);
            };

            request.onTransportFailure = function(errorMsg, request) {
                writeToScreen('Error occurred ' + errorMsg,
"systemError");
                if (window.EventSource) {
                    request.fallbackTransport = "sse";
                    transport = "see";
                }
                Service.startExercice(userId, exoId);
            };

            request.onReconnect = function(request, response) {
                writeToScreen('Connection lost, trying to reconnect.
Trying to reconnect ' + request.reconnectInterval, "systemError");
            };

            request.onReopen = function(response) {
                writeToScreen('Atmosphere re-connected using ' +
response.transport, "system");
            };

            request.onMessage = function(response) {
                var message = response.responseBody;
                try {
                    var json = atmosphere.util.parseJSON(message);
                } catch (e) {
                    logServer('This doesn\'t look like a valid JSON: '+
message, "WEBSOCKET MODULE");
                    return;
                }
                listener(json);
            };

            request.onClose = function(response) {
                clearInterval(mytimeout);
                if (response.state !== "unsubscribe") {
                    var current_request = {
                        type: "stop",
                        userId: userId,
                        exoId: exoId,
                        sessionId: broadcastId,
                        userUUID: userUUID,
                        courseId: courseId
                    };
                    var message =
atmosphere.util.stringifyJSON(current_request);
                    subSocket.push(message);
                }
            }
            subSocket = socket.subscribe(request);
        };


        function listener(data) {
            if (data.type === "stop") {
                socket.unsubscribe();
                broadcastId = '';
                subSocket = null;
```

```
                userUUID = 0;
                simulVariables = {};
                $(output).empty();
                $(consignes).empty();
                displayInstructions("remise à zéro", "stopMsg");
            } else if (data.type === "init") {
                launchSimulListener(data);
            } else if (data.type === "exoTip") {
                displayInstructions(data.message, "exoTipMsg");
            } else if (data.type === "initStep") {
                initSimul(data.message);
            } else if (data.type === "exoStart") {
                $(output).empty();
                $(consignes).empty();
                displayInstructions(data.message, "exoStartMsg");
            }
            else if (data.type === "validation") {
                if (data.inError) {
                    writeToScreen(data.message, "stepFailedMsg");
                } else {
                    writeToScreen(data.message, "stepSucceededMsg");
                }
            }
            else if (data.type === "stepTip") {
                displayInstructions(data.message, "stepTipMsg");
            } else if (data.type === "ctrlCheck") {
                if (data.inError) {
                    writeToScreen(data.message, "ctrlSucceededMsg");
                } else {
                    writeToScreen(data.message, "ctrlFailedMsg");
                }
            }
            else if (data.type === "endExo") {
                socket.unsubscribe();
                broadcastId = '';
                subSocket = null;
                userUUID = 0;
                simulVariables = {};
                $(output).empty();
                $(consignes).empty();
                displayInstructions(data.message, "exoSucceededMsg");
                reset();
            }
        }

        function makeRequest(userId, exoId, action) {
            var current_request = {
                type: action,
                userId: userId,
                exoId: exoId,
                sessionId: broadcastId,
                userUUID: userUUID,
                courseId: courseId
            };
            var message =
atmosphere.util.stringifyJSON(current_request);
            subSocket.push(message);
        }

        function sendState(userId, exoId, state) {
            var message = "";
```

```
            for (var key1 in state) {
                message += key1 + "=" + state[key1] + ";";
            }
            var current_request = {
                type: "state",
                userId: userId,
                exoId: exoId,
                sessionId: broadcastId,
                userUUID: userUUID,
                message: message,
                courseId: courseId
            };
            var message =
atmosphere.util.stringifyJSON(current_request);
            subSocket.push(message);
        }

        function writeToScreen(message, typeMsg) {
            if (message != "null"){
                $(output).append('<p class="' + typeMsg + '"
style="white-space: pre-wrap;word-wrap: break-word;word-break: break-
word;">' + message + '</p>');
                output.scrollTop = output.scrollHeight;
            }
        }

        function launchSimulListener(data) {
            logServer("Lancement de l'écoute de la simulation");
            var simulParam = atmosphere.util.parseJSON(data.message);
            simulVariables = simulParam.variables;
            if (simulVariables) {
                clearInterval(mytimeout);
                mytimeout = setInterval(getSimulState, 400);
            }
        }

        function getSimulState() {
            var currentState = {};
            if (simulVariables) {
                var circuitLoaded = false;
                for (var i = 0; i < simulVariables.length; i++) {
                    var result = _getVariable(simulVariables[i]);
                    currentState[simulVariables[i]] = result;
                    if (result){
                        circuitLoaded = true;
                    }
                }
                if (circuitLoaded && !(Object.deepEquals(oldState,
currentState))) {
                    sendState(userId, exoId, currentState);
                    oldState = currentState;
                }
            }
        }

        Service.startExercice = function(userId, exoId) {
            return makeRequest(userId, exoId, "start");
        };

        Service.validateExercice = function(userId, exoId) {
            return makeRequest(userId, exoId, "validate");
```

```
        };

        Service.stop = function(userId, exoId) {
            clearInterval(mytimeout);
            if (subSocket) {
                makeRequest(userId, exoId, "disconnect");
                socket.unsubscribe();
                broadcastId = '';
                subSocket = null;
                userUUID = 0;
                simulVariables = {};
            }
        };

        return Service;
    }]);
```

## G. Exemple de directive AngularJS

```
appFormid.directive('traceMonitoring', function($compile) {

    return {
        replace: true,
        restrict: 'E',
        //paramètres à passer à la directive via attributs dans la
balise HTML
        scope: {
            student: '=',
            exoId: '=',
            stepId: '=',
            getExo: '&',
            getStep: '&',
            getStepName: '&',
            showActivities: '=',
            getControlName: '&',
            hideValAction: '&',
            showValAction: '&',
            hideCtrlAction: '&',
            showCtrlAction: '&'
        },
        link: function($scope, $element, $attrs) {
            //méthodes utilitaires
            $scope.getLastTrace = function()
            {
                return _.last($scope.currentStep.tracesValidation);
            };
            $scope.isNotValidationTrace = function(trace) {
                return trace.idTypeTrace !== "V";
            };
            //Construction du template sous forme de string, puis
            // compilation pour interpréter les champs entre accolades
            $scope.buildTemplate = function() {
                var myTemplate = "";
                if ($scope.student) {
     var exo = $scope.getExo({student: $scope.student, exoId:
$scope.exoId});
     $scope.currentStep = $scope.getStep({exo: exo, stepId:
$scope.stepId});
     $scope.currentStepName = $scope.getStepName({exoId: $scope.exoId,
stepId: $scope.stepId});
     myTemplate += '<div class="stepData" ';
```

```
        myTemplate += ' ng-class="{ stepNeedHelp:currentStep.needHelp ==
true}">';
        myTemplate += '<div ng-show="student.isSelected"
class="stepStatut" ';
        myTemplate += '  tooltip-popup-delay="150" tooltip-append-to-
body="true" tooltip-html-unsafe="{{getLastTrace().contenu}}"';
        myTemplate += ' ng-class="{stepN: currentStep.result == 0,
stepF:currentStep.result == - 1, stepV:currentStep.result ==
1}"></div>';
        myTemplate += '<div ng-show="showActivities && student.isSelected"
ng-repeat="traceFormid in currentStep.traces |
filter:isNotValidationTrace"';
        myTemplate += ' tooltip-popup-delay="150" tooltip-append-to-
body="true" tooltip-html-unsafe="<p class=\'lead\'>
{{getControlName({trace :
traceFormid});}}</p>{{traceFormid.contenu}}"';
        myTemplate += ' ng-class="{ traceData:true,
traceN:traceFormid.resultat == 0, traceF:traceFormid.resultat == - 1,
traceV:traceFormid.resultat == 1}"></div></div>';
                var e = $compile(myTemplate)($scope);
                $element.replaceWith(e);
                    }
                };

                //reconstruction du template si un paramètre est modifié
                $scope.$watch("student", function() {
                    $scope.buildTemplate();
                });
                $scope.$watch("stepId", function() {
                    $scope.buildTemplate();
                });
                $scope.$watch("exoId", function() {
                    $scope.buildTemplate();
                });
                //gestion des actions de survol
                $scope.$on("show_popup", function(event, data) {
                    if (data.parent) {
                        if (data.parent.traceFormid) {
                            //cas du controle
                            $scope.showCtrlAction();
                        }
                        else {
                            //cas de la validation
                            $scope.showValAction();
                        }
                    }
                });
                //création de la trace à la disparition du popup
                $scope.$on("hide_popup", function(event, data) {
                    var params = {};
                    if (data.parent && (data.popupTime > 500)) {
                        params.data = {};
                        params.data.id_learner = $scope.student.id;
                        params.data.id_exercise = $scope.exoId;
                        params.data.id_step = $scope.stepId;
                        params.data.popup_time = (data.popupTime / 1000);
                        if (data.parent.traceFormid) {
                            //cas du controle
                            params.data.id_ctrl =
data.parent.traceFormid.controlek;
```

```
                        params.data.simulInfo =
data.parent.traceFormid.contenu;
                        params.data.ctrl_rank = data.parent.$index + 1;
                        params.data.color =
((data.parent.traceFormid.resultat == 1) ? "GREEN" :
((data.parent.traceFormid.resultat == -1) ? "RED" : "ORANGE"));
                        $scope.hideCtrlAction(params);
                    }
                    else {
                        //cas de la validation
                        var trace = $scope.getLastTrace();
                        if (trace) {
                            params.data.simulInfo = trace.contenu;
                            params.data.val_rank = 1;
                            params.data.color =
(($scope.currentStep.result == 1) ? "GREEN" :
(($scope.currentStep.result == -1) ? "RED" : "ORANGE"));
                        } else
                        {
                            params.data.val_rank = 0;
                        }
                        $scope.hideValAction(params);
                    }
                }
            });

            $scope.buildTemplate();
        }
    };
});
```

## H. Rôle et sécurité : utilisation du calcul binaire

```
(function(exports){

    var userRoles = {
        public: 1, // 001
        user:   2, // 010
        admin:  4  // 100
    };

    exports.userRoles = userRoles;
    exports.accessLevels = {
        student: userRoles.student | // 111
                 userRoles.tutor   |
                 userRoles.admin,
        tutor:   userRoles.tutor,  // 001
        author: userRoles.author |   // 110
                 userRoles.admin,
        admin:  userRoles.admin    // 100
    };
})(typeof exports === 'undefined'? this['routingConfig']={}: exports);

angular.module('myApp', ['myApp.services', 'ngCookies'])
.config(['$routeProvider', '$locationProvider',
    function ($routeProvider, $locationProvider) {

    // ...

    var access = routingConfig.accessLevels;
```

```
    $routeProvider.when('/author',
                        {
                            templateUrl: 'partials/authorView.html',
                            controller: 'AuthorCtrl',
                            access: access.author //accès binaire
                        });
    $routeProvider.when('/tutor',
                        {
                            templateUrl: 'partials/tutorView.html',
                            controller: 'TutorCtrl',
                            access: access.tutor
                        });
    $routeProvider.when('/admin',
        {
            templateUrl:    'partials/admin',
            controller:     AdminCtrl,
            access:         access.admin
        });

    // ...

}]);

angular.module('angular-client-side-auth')
.factory('Auth', function($http, $rootScope, $cookieStore){

    // ...

    $rootScope.accessLevels = accessLevels;
    $rootScope.userRoles = userRoles;

    return {
        authorize: function(accessLevel, role) {
            if(role === undefined)
                role = $rootScope.user.role;
            return accessLevel & role; //comparaison binaire
        },

        isLoggedIn: function(user) {
            if(user === undefined)
                user = $rootScope.user;
            return user.role === userRoles.user || user.role ===
userRoles.admin;
        },

        register: function(user, success, error) {
            $http.post('/register',
user).success(success).error(error);
        },

        login: function(user, success, error) {
            $http.post('/login', user).success(function(user){
                $rootScope.user = user;
                success(user);
            }).error(error);
        },

        logout: function(success, error) {
            $http.post('/logout').success(function(){
                $rootScope.user = {
                    username = '',
```

```
                    role = userRoles.public
                };
                success();
            }).error(error);
        },

        accessLevels: accessLevels,
        userRoles: userRoles
    };
});
```

# I. Fichier de spécification des traces tuteurs

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA | AB |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | learner | | | | step | | | | learner/step | | details | |
| 2 | | global | time gap | id learner | info selected | global prog | info step | prev step | last step | ctrlId | rang_Val | rang_Ctrl | color | simul_info | seq_val | seq_ctrl | targeted | | list selected | | targeted | | list displayed | | targeted | List selected | val | ctrl |
| 3 | | | | | | | | | | | | | | | | | Global | Detail | Global | Detail | Global | Detail | Global | Detail | Global | Global | | |
| 4 | open_session | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 5 | manual_request | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 6 | auto_request_start | X | X | | | | | | | | | | | | | | | | | | | | | | | | | |
| 7 | auto_request_stop | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 8 | adjust_time_gap | X | X | | | | | | | | | | | | | | | | | | | | | | | | | |
| 9 | quit_session | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | N1_Hover_Validation | X | | X | | | X | | | | X | | X | X | | | X | | | | | | | | X | | | |
| 11 | N1_Hover_Control | X | | X | | | X | | | X | | X | X | X | | | | | | | | | | | X | | | |
| 12 | N1_Select_all_learners | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 13 | N1_Unselect_all_learners | X | | | X | | | | | | | | | | | | | | | X | | | | | | | | |
| 14 | N1_Unselect_one_learner | X | | X | X | | | | | | | | | | | | X | | | X | | | | | | | | |
| 15 | N1_Add_learner_to_selection | X | | X | X | | | | | | | | | | | | X | | | X | | | | | | | | |
| 16 | N1_Select_one_learner | X | | X | X | | | | | | | | | | | | X | | | | | | | | | | | |
| 17 | N1_View_all_exercises | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 18 | N1_Choose_Step | X | | | X | | X | | | | | | | | | | | | | | X | | X | | | | | |
| 19 | N1_Scroll_Right | X | | | X | | X | | X | | | | | | | | | | | | | | | | | | | |
| 20 | N1_Scroll_Left | X | | | X | | X | | X | | | | | | | | | | | | | | | | | | | |
| 21 | N1_Hide_controls | X | | | X | X | | | | | | | | | | | | | | | | | | | | | | |
| 22 | N1_Show_controls | X | | | X | X | | | | | | | | | | | | | | | | | | | | | | |
| 23 | N2_Hover_Validation | X | | X | | | X | | | | X | | X | X | X | | | | | | | | | | X | | X | |
| 24 | N2_Hover_Control | X | | X | | | X | | | X | | X | X | X | | X | | | | | | | | | X | | | X |
| 25 | N2_Select_all_learners | X | | | | | X | | | | | | | | | | | | | | | | | | | | | |
| 26 | N2_Unselect_all_learners | X | | | X | | X | | | | | | | | | | | | | | | | | X | | | | |
| 27 | N2_Unselect_one_learner | X | | X | X | | X | | | | | | | | | | | | | | | | | X | X | | | |
| 28 | N2_Add_learner_to_selection | X | | X | X | | X | | | | | | | | | | | | | | | | | X | X | | | |
| 29 | N2_Select_one_learner | X | | X | X | | X | | | | | | | | | | | | | | | | | | X | | | |
| 30 | N2_Choose_Step | X | | | X | | X | X | | | | | | | | | | | | | | | | | | | | |
| 31 | N2_View_all_exercises | X | | | X | | | X | | | | | | | | | | | | | | | | | | | | |
| 32 | N2_Scroll_Right | X | | | X | | X | | X | | | | | | | | | | | | | | | | | | | |
| 33 | N2_Scroll_Left | X | | | X | | X | | X | | | | | | | | | | | | | | | | | | | |
| 34 | N2_Next_Ctrl | X | | | X | | X | | | | | | | | | | | | | | | | | | | | | |
| 35 | N2_Prev_Ctrl | X | | | X | | X | | | | | | | | | | | | | | | | | | | | | |
| 36 | N3_Hover_Validation | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 37 | N3_Hover_Control | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 38 | N3_Select_all_learners | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 39 | N3_Unselect_all_learners | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 40 | N3_Unselect_one_learner | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 41 | N3_Add_learner_to_selection | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 42 | N3_Select_one_learner | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 43 | N3_Choose_Step | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 44 | N3_View_all_exercises | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 45 | N3_Scroll_Right | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 46 | N3_Scroll_Left | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 47 | N3_Next_Ctrl | X | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 48 | N3_Prev_Ctrl | X | | | | | | | | | | | | | | | | | | | | | | | | | | |

Détail des paramètres complexes :

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | global | | selected_learner_info | | global progression | |
| 2 | action_timestamp | | nb_selected_learners | | nb_validations | |
| 3 | userId | | list_selected_learners | | nb_controls | |
| 4 | action | | | | | |
| 5 | start_level | | | | | |
| 6 | end_level | | | | progression | |
| 7 | action_counter | | | | nb_total | |
| 8 | time_btw_2_actions | | | | nb_green | |
| 9 | sessionId | | | | nb_red | |
| 10 | | | | | nb_orange | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | step_info | | learner global progression | | step global progression | |
| 15 | exerciseId | | learnerId | | step_info | |
| 16 | stepId | | progression | | progression | |
| 17 | | | | | | |
| 18 | control global progression | | learner detail progression | | step detail progression | |
| 19 | step_info | | learnerId | | step_info | |
| 20 | ctrlId | | liste de step progression | | liste de learner progression | |
| 21 | progression | | | | | |
| 22 | | | | | | |
| 23 | | | | | | |
| 24 | selected learners global progression | | learner step progression | | validation global info | |
| 25 | liste de learner progression | | learnerId | | step_info | |
| 26 | | | step_info | | progression | |
| 27 | selected learners detail progression | | progression | | | |
| 28 | liste de learner/step progression | | | | | |
| 29 | | | | | ctrl global info | |
| 30 | displayed steps global progression | | | | step_info | |
| 31 | liste de step progression | | | | ctrlId | |
| 32 | | | | | progression | |
| 33 | | | | | | |

## J. Fichier de configuration du serveur applicatif

Uniquement les passages paramétrés pour FORMID

```
[…]
<profile>
   <subsystem xmlns="urn:jboss:domain:logging:2.0">
      <!—Paramétrage des traces console -->
      <console-handler name="CONSOLE">
         <level name="INFO"/>
         <formatter>
            <pattern-formatter pattern="%K{level}%d{HH:mm:ss,SSS} %-5p
[%c] (%t) %s%E%n"/>
         </formatter>
      </console-handler>
      <!—Paramétrage des traces seveur -->
      <periodic-rotating-file-handler name="FILE" autoflush="true">
         <level name="DEBUG"/>
         <formatter>
            <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c] (%t)
%s%E%n"/>
         </formatter>
         <file relative-to="jboss.server.log.dir" path="server.log"/>
         <suffix value=".yyyy-MM-dd-HH"/>
         <append value="true"/>
      </periodic-rotating-file-handler>
      <!Paramétrage des traces FORMID -->
      <periodic-rotating-file-handler name="NEEDTRACE"
autoflush="true">
         <level name="DEBUG"/>
         <formatter>
            <pattern-formatter pattern="%d{HH:mm:ss,SSS} %-5p [%c]
%s%E%n"/>
         </formatter>
         <file relative-to="jboss.server.log.dir" path="formid.log"/>
         <suffix value=".yyyy-MM-dda"/>
         <append value="true"/>
      </periodic-rotating-file-handler>
[…]
      <!-- Attribution du log « FILE » au serveur web -->
      <logger category="io.undertow">
         <level name="FINEST"/>
         <handlers>
             <handler name="FILE"/>
           </handlers>
         </logger>
         <!-- Attribution du log « FILE » au serveur web -->
         <logger category="fr.imag.lig.metah">
            <level name="FINEST"/>
            <handlers>
               <handler name="NEEDTRACE"/>
            </handlers>
         </logger>
[…]
        </subsystem>
[…]
        <subsystem xmlns="urn:jboss:domain:datasources:2.0">
            <datasources>
                <datasource jndi-
name="java:jboss/datasources/FormID_DS" pool-name="FormID_DS"
enabled="true" use-java-context="true">
```

```
                    <connection-url>
jdbc:h2:tcp://localhost/formidDB/formidData</connection-url>
                    <driver>h2</driver>
                    <security>
                        <user-name>sa</user-name>
                    </security>
                </datasource>
                <drivers>
                    <driver name="h2" module="com.h2database.h2">
                        <xa-datasource-class>
org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
                    </driver>
                </drivers>
            </datasources>
        </subsystem>
[…]
```

MEMOIRE D'INGENIEUR C.N.A.M. en Informatique option Systèmes d'Information (ISI)

# Évolution de la suite logicielle FORMID pour la conception, l'exécution et le suivi de situations d'apprentissage à distance

**Patrick ECHTERBILLE** **Grenoble, le 03 Juin 2013**

---


## Résumé

Ce document retrace le travail effectué dans le cadre du stage de fin de cycle d'ingénieur CNAM. Ce stage s'est déroulé dans l'équipe MeTAH du Laboratoire Informatique de Grenoble, autour de la suite logicielle FORMID. Cette suite composée de trois modules est dédiée à la conception, l'exécution et le suivi de situations d'apprentissage à distance.

Il y est décrit les choix de changements technologiques, entrainant une migration d'applets Java vers une architecture client-serveur utilisant les dernières technologies web. En plus de la correction des anomalies liées à la sécurité et à la compatibilité de l'application, il relate la mise en œuvre des moyens d'industrialisation et d'internationalisation de la suite FORMID. Pour finir, le document décrit la modélisation et l'implémentation d'un nouveau système de traçage avec des indicateurs de qualité et l'extension du module de conception ainsi que sa connexion à une ontologie de représentation des connaissances.


**Mots-clés : EIAH, scénario pédagogique, praxéologie, architecture client-serveur, AngularJS, traces**

---


## Abstract

This paper describes the work done in the internship of CNAM engineer cycle. This one was held in MeTAH team Grenoble Informatics Laboratory, around the FORMID suite. This suite consists of three modules which are dedicated to the design, implementation and monitoring of distance learning situations.

It is described the choice of technological change, resulting in a migration of Java applets to a client-server architecture using the latest web technologies. In addition to the correction of anomalies related to security and application compatibility, it describes the implementation of FORMID industrialization and internationalization. Finally, the paper describes the modeling and implementation of a new tracking system with indicators of quality and the design module extension and its connection to an ontology of knowledge representation .


**Keywords : TEL, pedagogical scenario, praxeology, client-server model, AngularJS, log**